Escuela Superior de Ingeniería

Programa de Doctorado en Ingeniería y Arquitectura

TESIS DOCTORAL CON MENCIÓN INTERNACIONAL

Desarrollo Dirigido por Modelos de Interfaces Específicas de Dominio para el Procesamiento de Eventos Complejos en Arquitecturas Orientadas a Servicios

Model-Driven Development of Domain-Specific Interfaces for Complex Event Processing in Service-Oriented Architectures

Autor

Juan Boubeta Puig

Directoras

Dra. Guadalupe Ortiz Bellot y Dra. Inmaculada Medina Bulo

Cádiz, julio de 2014

Conformidad de las Directoras

Dª Guadalupe Ortiz Bellot, Profesora Titular de Universidad del Departamento de Ingeniería Informática de la Universidad de Cádiz, y Dª María Inmaculada Medina Bulo, Titular de Universidad del Departamento de Ingeniería Informática de la Universidad de Cádiz, siendo Directoras de la Tesis titulada *Desarrollo Dirigido por Modelos de Interfaces Específicas de Dominio para el Procesamiento de Eventos Complejos en Arquitecturas Orientadas a Servicios*, realizada por el doctorando D. Juan Boubeta Puig dentro del Programa de Doctorado en Ingeniería y Arquitectura, para proceder a los trámites conducentes a la presentación y defensa de la tesis doctoral arriba indicada, informan que se autoriza la tramitación de la tesis.

Las directoras de tesis

Guadalupe Ortiz Bellot

María Inmaculada Medina Bulo

En Cádiz, España, a 12 de junio de 2014

Agradecimientos

Quisiera manifestar mis agradecimientos a todas las personas que de una u otra forma han hecho posible que esta tesis doctoral vea la luz:

- A mis padres, por su ejemplo, actitud de escucha y apoyo incondicional, confianza y cariño.
- A mi hermana Alejandra, por su tiempo, dedicación y todos los momentos compartidos durante la elaboración de esta tesis.
- A mi hermana Priscila, mi cuñado Javi y mi sobrina Martina, así como al resto de miembros de mi familia, por creer en mí y los buenos momentos compartidos.
- A Lupe e Inma, por su gran dedicación, paciencia, apoyo y buenos consejos para que esta tesis pudiese llegar a buen puerto.
- A Martin Kappes, por darme la oportunidad de realizar la estancia de investigación en Frankfurt, y a su grupo de investigación, especialmente a Ruediger Gad y Robin Müller-Bady por su ayuda e implicación en la evaluación de esta tesis.
- A Mike Papazoglou, por darme la oportunidad de realizar la estancia de investigación en Tilburg, y a todos los investigadores y amigos que conocí allí y de los que conservo muy buenos recuerdos.
- A Víctor Ayllón y Juan M. Reina, por darme a conocer el procesamiento de eventos complejos y hacerme partícipe de sus experiencias en esta tecnología.
- A los miembros de mi grupo de investigación UCASE, especialmente a Antonio por introducirme en el mundo de Epsilon.
- A mis compañeros del Departamento de Ingeniería Informática, por sus ánimos.
- A mis amigos, por su paciencia, escucha y por estar siempre ahí en los buenos y en los malos momentos.

Muchas gracias a todos.

Juan Boubeta Puig

Agradecimientos Institucionales

Este trabajo ha sido financiado por el proyecto MoDSOA (TIN2011-27242) del Programa Nacional de Investigación, Desarrollo e Innovación del Ministerio de Ciencia e Innovación y por el proyecto PR2011-004 del Plan de Promoción de la Investigación de la Universidad de Cádiz.

Resumen

En la actualidad, las empresas y organizaciones de todo el planeta necesitan gestionar cada día una ingente cantidad de datos provenientes de fuentes muy diversas, tales como aplicaciones propias y de terceros, servicios web, sensores, plataformas de Internet de las cosas o redes sociales, con el fin de llevar a cabo la toma de decisiones.

Un buen proceso de toma de decisiones requiere, entre otros factores, conocer tan pronto como sea posible cuál es el valor que tienen dichos datos para el negocio empresarial. La realización de un exhaustivo análisis de datos, así como una actuación temprana en relación a las situaciones críticas o relevantes que supongan una amenaza para la empresa, permitirá que esta se posicione por encima de sus competidores. No obstante, se trata de un proceso bastante complejo, debido, entre otras razones, a que gran parte de estos datos son heterogéneos —no comparten un formato común— y además deberían procesarse en tiempo real.

En este contexto, el procesamiento de eventos complejos o CEP (*Complex Event Processing*) es una de las tecnologías software que permite analizar y correlacionar grandes volúmenes de datos en forma de eventos con el propósito de detectar situaciones de una mayor complejidad semántica, así como inferir conocimiento valioso que ayudará en el proceso de toma de decisiones. Para ello, se hace uso de los denominados patrones de eventos en los que se especifican las condiciones que han de cumplirse para detectar dichas situaciones de interés.

A pesar de las grandes ventajas que CEP puede aportar en el ámbito empresarial, supone un gran reto para los usuarios que son expertos en el negocio, pero que carecen de la experiencia y los conocimientos necesarios para el uso de esta tecnología. Uno de los principales problemas a los que deben enfrentarse estos usuarios es precisamente la definición de dichos patrones usando algún lenguaje concreto, los denominados lenguajes de procesamiento de eventos o EPL (*Event Processing Language*). Aunque algunas soluciones software actuales ofrecen herramientas gráficas con la finalidad de solventar este problema, estas no son lo suficientemente amigables, puesto que requieren que el usuario escriba manualmente, al menos, una parte del código necesario para la definición de los patrones.

Por otra parte, los sistemas de información actuales tienden a estar basados en arquitecturas orientadas a servicios o SOA (*Service-Oriented Architecture*), debido a que este tipo de arquitectura software permite desarrollar sistemas distribuidos altamente escalables e integrables con otros sistemas propios o de terceros. Recientemente, estas se están combinando con las arquitecturas dirigidas por eventos o EDA (*Event-Driven Architectu-*

re) dando como resultado las denominadas arquitecturas orientadas a servicios y dirigidas por eventos —ED-SOA (*Event-Driven Service-Oriented Architecture*) o SOA 2.0—, que se caracterizan porque las comunicaciones entre los usuarios, las aplicaciones y los servicios se realizan por medio de eventos de una forma totalmente desacoplada. Para detectar, en tiempo real, situaciones críticas o relevantes en estos sistemas complejos y heterogéneos, así como de ejecutar las acciones pertinentes, se hace necesaria la integración de CEP con SOA 2.0.

Con el fin de dar respuesta a estas necesidades, la investigación llevada a cabo en esta tesis doctoral se ha centrado en el desarrollo dirigido por modelos de interfaces específicas de dominio para CEP en SOA 2.0, con el objeto de facilitar a los expertos en el negocio la definición tanto de los patrones que necesiten detectar en sus sistemas de información, como de las alertas que deban notificarse en tiempo real. Para lograrlo, se ha propuesto un enfoque dirigido por modelos para CEP en SOA 2.0, un lenguaje de modelado y un editor gráfico para la definición de dominios CEP, un lenguaje de modelado y un editor gráfico reconfigurable para la definición y la generación de código de patrones de eventos, así como una solución tecnológica que integra CEP con SOA 2.0. Para la evaluación del enfoque se han desarrollado dos casos de estudio que confirman que este es independiente del dominio donde se aplique. Asimismo, se concluye que los lenguajes definidos son independientes del código que implemente los patrones de eventos y las acciones a llevar a cabo, así como que los editores gráficos facilitan todo esto de una forma amigable e intuitiva, y haciendo los detalles de implementación totalmente transparentes para los expertos en el negocio. Se trata, por tanto, de un enfoque novedoso que pone la tecnología CEP al alcance de cualquier usuario, repercutiendo beneficiosamente en el proceso de toma de decisiones.

Abstract

Nowadays, companies and organizations all around the world need to manage huge amounts of data from heterogeneous sources —such as own and third-party applications, web services, sensors, Internet of things platforms or social networks— every day in order to conduct the decision-making process.

To be successful, a decision-making process requires, among other factors, prompt information regarding what the value of such data is for the business in question. A thorough analysis of data, as well as early action for critical or relevant situations considered as threats for an organization, will allow the organization to be positioned above its competitors. Nevertheless, it is a complex process since, among other reasons, data are heterogeneous —they do not share a common format— and they should be processed in real time.

In this context, Complex Event Processing (CEP) is a technology that allows the analysis and correlation of large volumes of data with the aim of detecting complex and meaningful events, and of inferring valuable knowledge for end users. This knowledge will be really helpful in the decision-making process. To do this, so-called event patterns are used. These patterns specify which conditions must be met in order to detect such situations of interest.

Despite the great advantages that CEP can bring to a business, it is a substantial challenge for users who are business experts, but do not have the necessary experience and knowledge for the use of this technology. One of the main problems these users have to face is precisely the definition of these event patterns using particular languages, those called Event Processing Languages (EPLs). Although some current software solutions provide graphical tools in order to solve this problem, none of them are user-friendly enough, since they require users to hand-write at least a portion of the code for the pattern definition.

On the other hand, current information systems tend to be based on Service-Oriented Architectures (SOAs) due to the fact that this type of software architecture allows the development of highly scalable distributed systems as well as their integration with own and third-party systems. Recently, these are being combined with Event-Driven Architectures (EDAs), what is known as Event-Driven Service-Oriented Architectures (ED-SOAs or SOAs 2.0). The latter enable us to establish decoupled communications between users, applications and services. The integration of CEP with SOA 2.0 is a requirement to be able to detect real-time, relevant or critical situations in these complex and heterogeneous systems, as well as for the execution of the appropriate actions.

In order to respond to these needs, the research carried out in this thesis has focused on the model-driven development of domain-specific interfaces for CEP in SOAs 2.0, with the aim of facilitating the task of defining both event patterns to be detected and alerts for real time notification for domain experts. To reach this goal, the following contributions have been supplied: a model-driven approach for CEP in SOA 2.0, a modeling language and a graphical editor for CEP domain definition, a modeling language and a reconfigurable graphical editor for event pattern definition and code generation, as well as a technological solution integrating CEP with SOA 2.0. The approach has been evaluated through the development of two case studies that have confirmed independence of the approach from the application domain. Besides, we can assert that the proposed languages are independent of event patterns and actions implementation code and that user-friendly and intuitive graphical editors hide all the implementation details from end users. This is therefore a novel approach for bringing CEP technology closer to any user, positively impacting the decision making process.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Listados de Código

Capítulo 1

Introducción

«Investigar es ver lo que todo el mundo ha visto, y pensar lo que nadie más ha pensado.»
(Albert Szent-Györgi)

En este capítulo se describe la motivación que ha llevado a realizar esta tesis doctoral y se detallan los objetivos de la misma. Además, se especifican las principales contribuciones, así como la estructura del resto de la disertación.

1.1. Motivación

En la actualidad, fluyen diariamente enormes volúmenes de datos por una gran cantidad de sistemas de información heterogéneos y distribuidos a nivel mundial [Tsu12]. Con el fin de dirigir satisfactoriamente las decisiones y/o acciones en el entorno empresarial, los expertos del negocio, habrán de obtener y gestionar eficientemente toda esta información en tiempo real para poder descubrir situaciones relevantes en su ámbito empresarial [Han13].

En este sentido, *big data* es un enfoque emergente cuya misión principal es la gestión y el análisis de esta ingente cantidad de datos, que no puede ser procesada por herramientas software tradicionales, con la intención de convertir estos datos en información valiosa como soporte para la toma de decisiones. *Big data* es reconocido por su modelo denominado de las tres uves: volumen, velocidad y variedad [Rus11]. El volumen hace referencia a la cantidad de datos que pueden ser manipulados y almacenados cada día. La velocidad es la dimensión que trata de analizar cuán rápido pueden ser obtenidos y analizados estos datos. Por otro lado, la variedad alude a los distintos tipos de datos que pueden ser gestionados, algunos menos pesados como el tipo texto, y otros que requieren mayor capacidad de almacenamiento como pueden ser el audio y el vídeo, entre otros.

Sin embargo, este enfoque se centra fundamentalmente en datos que son obtenidos a priori y almacenados en bases de datos. Por esta razón, esta podría no ser la mejor solución para procesar datos de naturaleza muy diversa y provenientes de fuentes de datos heterogéneas en tiempo real. Para solucionarlo, *big data* puede complementarse con *fast*

data [Han13], un enfoque que precisamente permite analizar los datos continuamente, tratando de determinar por qué ciertos datos suponen un valor añadido para el negocio. A esta nueva dimensión se le conoce como la cuarta uve de dicho modelo: el valor que tienen los datos para el negocio.

Con el propósito de detectar situaciones relevantes o críticas en el negocio, *fast data* permite utilizar tecnologías dinámicas como el procesamiento de eventos complejos o CEP (*Complex Event Processing*) [Luc02], una tecnología software que utiliza un conjunto de técnicas para hacer un uso más eficiente de las arquitecturas dirigidas por eventos o EDA (*Event-Driven Architecture*) [Tay08]. CEP permite procesar y analizar grandes cantidades de eventos, así como correlacionarlos para detectar y responder en tiempo real a situaciones críticas del negocio. Para ello, las condiciones que describen las situaciones que se pretenden detectar se especifican usando unas plantillas especiales, denominadas patrones de eventos. Estos patrones de eventos serán añadidos a un motor CEP, el software que se encargará tanto de analizar y correlacionar los eventos provenientes de distintas fuentes de información, como de generar los eventos complejos tras la detección de estos patrones, indicando cuáles han sido las situaciones acontecidas. Dicho de otro modo, estos patrones inferirán nuevos eventos más complejos y con un mayor significado semántico, que ayudarán a tomar decisiones tan pronto como sea posible.

Además, estos patrones de eventos son definidos utilizando lenguajes específicos desarrollados para este propósito, conocidos como EPL (*Event Processing Language*). Sin embargo, se necesita un gran conocimiento en este tipo de lenguajes para proceder a la definición de dichos patrones. Por consiguiente, uno de los grandes inconvenientes del uso de CEP por parte de usuarios no tecnólogos —justamente los usuarios que poseen un vasto conocimiento del dominio específico donde los patrones deben ser detectados— es la gran curva de aprendizaje requerida para convertirse en expertos de estos lenguajes. Algunas soluciones software como Oracle CEP Visualizer [Ora14], StreamBase Studio [Str14] y SAP Sybase ESP Studio [Syb14], ofrecen herramientas gráficas con la finalidad de solventar este problema. No obstante, estas herramientas requieren aún a los inexpertos en CEP que escriban a mano, al menos, una parte del código EPL que implementa al patrón de eventos en cuestión.

En resumen, CEP es una tecnología que permite pronosticar un avance en la prevención y detección de situaciones críticas en tiempo real, aunque no proporciona aún interfaces lo suficientemente amigables para que un usuario no tecnólogo pueda beneficiarse de su gran utilidad. Esta situación conlleva a que estos usuarios tengan que depender continuamente de expertos en software que les ofrezcan soporte técnico para lograr interactuar con interfaces que son poco intuitivas o difíciles de utilizar.

Como solución a esta problemática, esta tesis doctoral, denominada *Desarrollo Dirigido por Modelos de Interfaces Específicas de Dominio para el Procesamiento de Eventos Complejos en Arquitecturas Orientadas a Servicios*, persigue la creación de interfaces específicas de dominio, para facilitar al usuario final la definición de los patrones de eventos a detectar y de las acciones a llevar a cabo cuando se detecten estos patrones de manera gráfica e intuitiva; además, mediante el uso de técnicas de desarrollo de software dirigido por modelos o MDD (*Model-Driven Development*) [Sta06] se integrarán las decisiones de

negocio definidas por el usuario en la interfaz con los motores de detección de eventos y los sistemas del cliente en cuestión. El MDD se centra en el desarrollo de aplicaciones basadas en modelos en lugar de en tecnologías y en la transformación entre modelos y de modelo a código con el propósito de mejorar la productividad así como el mantenimiento de los sistemas software. Estos modelos se expresan a través de los denominados lenguajes de modelado específicos de dominio o DSML (*Domain-Specific Modeling Language*) [GM13a].

Los sistemas del cliente en la actualidad tienden a estar basados en arquitecturas orientadas a servicios o SOA (*Service-Oriented Architecture*) [Pap07a]; por lo cual, para poder detectar estos eventos en sistemas complejos y heterogéneos será necesario integrar CEP con SOA. A la integración de EDA y SOA se le denomina arquitectura orientada a servicios y dirigida por eventos o ED-SOA (*Event-Driven Service-Oriented Architecture*), más conocida recientemente como SOA 2.0. Esta última es una evolución de las arquitecturas orientadas a servicios tradicionales en la que la comunicación entre los usuarios y los servicios se realiza en base a eventos; esto es, los servicios son desencadenados por eventos y reaccionan a estos, así como pueden ser fuente de nuevos eventos de salida [Luc12]. La combinación de los paradigmas EDA y SOA es posible gracias al uso de un ESB (*Enterprise Service Bus*), un *middleware* que facilita la integración de diferentes aplicaciones y servicios de entornos heterogéneos proporcionando interoperabilidad entre los distintos protocolos de comunicación [Dav09], permitiendo además la exposición de las distintas aplicaciones como servicios.

Así pues, el usuario final —experto en salud, bolsa y mercados, seguridad informática o domótica, entre otros— podrá ser informado de los distintos eventos de interés —eventos complejos— tan pronto como ocurran en los sistemas de información de su compañía u organización. Para lograrlo, simplemente deberá haber modelado previamente, a través de una interfaz intuitiva y amigable, los patrones de eventos sobre las situaciones críticas o relevantes en las que esté interesado. Es importante destacar que el saber qué está pasando justo ahora, repercutirá positivamente en la toma de decisiones empresariales.

1.2. Objetivos

Esta tesis doctoral satisface los siguientes objetivos:

- **Objetivo 1**: Recopilar el estado del arte de los enfoques existentes para CEP, los editores gráficos para la definición y la generación de código de patrones de eventos, así como las propuestas para la integración de CEP con EDA y/o SOA.

  Para alcanzar este objetivo, se realizará una exhaustiva revisión bibliográfica sobre dichos aspectos.

- **Objetivo 2**: Definir un enfoque dirigido por modelos que haga posible el procesamiento de eventos complejos en SOA 2.0, minimizando la curva de aprendizaje en las tecnologías requeridas para la detección de situaciones críticas o relevantes en tiempo real por parte del usuario final.

Para ello, se determinarán las fases de las que debe estar compuesto este enfoque, tanto en tiempo de diseño como de ejecución. Así el enfoque será independiente tanto del dominio donde deba usarse CEP, como de los EPL requeridos para implementar los patrones e independiente también de los lenguajes para detallar las acciones a llevar a cabo.

- **Objetivo 3**: Definir un DSML y construir un editor gráfico de dominios CEP que faciliten a cualquier usuario, experto en un dominio concreto pero no en CEP, la descripción de los tipos de eventos y propiedades característicos de dicho dominio.

  Para alcanzar este objetivo, se analizará cómo llevar a cabo la definición del DSML y el desarrollo del editor gráfico con el propósito de poner al alcance de cualquier usuario no tecnólogo el modelado de estos dominios.

- **Objetivo 4**: Definir un DSML y construir un editor gráfico de patrones de eventos que permitan expresar mediante modelos gráficos e intuitivos qué situaciones se desean detectar para un dominio en particular y si estas deben ser notificadas a los usuarios interesados mediante algún servicio como, por ejemplo, de correo electrónico o de redes sociales.

  Para ello, el editor gráfico se adaptará al contexto específico en el que se utilice, reconfigurándose a partir del dominio CEP modelado por el experto en el dominio. Además, el editor deberá transformar estos modelos, mediante el uso de técnicas de MDD, a código entendible tanto por el motor CEP donde se desplegará el patrón, como por la SOA 2.0 donde se ejecutarán las acciones que se necesite llevar a cabo tras la detección del patrón. Para satisfacer este objetivo, se estudiarán las diversas alternativas para la definición del DSML y de los modelos con los que el usuario no experto en tecnologías va a poder expresar de forma sencilla los patrones de eventos para un dominio en particular, así como describir, de forma transparente para él, los servicios a invocar cuando se detecten dichos patrones. Asimismo, se estudiarán los distintos marcos de trabajo existentes para la creación de editores gráficos de modelado, optándose por el que permita implementar interfaces amigables, intuitivas y reconfigurables, y se definirán las reglas de validación y transformación de dichos modelos.

- **Objetivo 5**: Integrar CEP con SOA 2.0, así como con los editores gráficos de modelado para hacer realidad el enfoque dirigido por modelos propuesto.

  Para lograr esta integración de CEP con SOA 2.0, se hará uso de un ESB, como nexo de unión, y se definirán dos casos de estudio representativos de dicha integración para comprobar la versatilidad del enfoque propuesto.

- **Objetivo 6**: Evaluar el enfoque dirigido por modelos y los DSML propuestos,

así como la funcionalidad y usabilidad de los editores gráficos.

Para ello, se analizará si los DSML son independientes del EPL que implemente los patrones de eventos, se probarán y evaluarán los editores por usuarios finales y, finalmente, se estudiará el incremento de productividad al hacer uso de dicho enfoque.

1.3. Contribuciones

En esta sección, se enumeran las principales contribuciones derivadas del trabajo desarrollado en esta tesis doctoral:

1. Un estado del arte sobre los enfoques existentes para CEP, los editores gráficos para la definición y la generación de código de patrones de eventos, así como las propuestas para la integración de CEP con EDA y/o SOA (véase el Capítulo 2).

2. Un enfoque dirigido por modelos para el procesamiento de eventos complejos en SOA 2.0 (véase el Capítulo 3).

3. Un DSML y un editor gráfico para la definición de dominios CEP (véase el Capítulo 4).

4. Un DSML y un editor gráfico reconfigurable para la definición y la generación de código de patrones de eventos (véase el Capítulo 5).

5. Una solución tecnológica para la integración del procesamiento de eventos complejos con una SOA 2.0 (véase el Capítulo 6). Esta solución hace posible la obtención de datos desde fuentes heterogéneas, el análisis y la correlación de la información en tiempo real para la detección de situaciones críticas o relevantes para un dominio concreto, así como la generación de alarmas o avisos ante la ocurrencia de los patrones definidos y su notificación a los usuarios interesados.

Téngase en cuenta que estas contribuciones serán descritas pormenorizadamente en los capítulos mencionados.

1.4. Estructura de la Tesis

A continuación se describe brevemente el contenido de los capítulos en los que se estructura esta memoria de tesis doctoral. Cabe destacar que la solución tecnológica, que se integra con los editores, no se explica hasta el Capítulo 6, por haberse considerado que la explicación inicial del enfoque dirigido por modelos y los siguientes capítulos con los DSML y los editores, facilitarían la comprensión del anterior.

- En este Capítulo 1 se especifica la motivación de este trabajo, se enumeran los principales objetivos de esta tesis, así como las principales contribuciones.

- El Capítulo 2 presenta una recopilación de los conceptos fundamentales sobre MDD, SOA, EDA, SOA 2.0 y CEP relacionados con las tecnologías y marcos de trabajo utilizados durante el desarrollo de esta tesis doctoral; además, se detallan los principales trabajos relacionados.

- En el Capítulo 3 se propone un enfoque dirigido por modelos para el procesamiento de eventos complejos en SOA 2.0, describiendo los pasos que deben seguirse desde que el experto en el dominio modela un dominio CEP hasta que se notifican al usuario final, en tiempo real, las situaciones críticas en las que esté interesado; previo modelado de los patrones de eventos correspondientes.

- El Capítulo 4 motiva y describe el DSML para el modelado de dominios CEP con el propósito de unificar la descripción de estos dominios mediante el uso de modelos y de abstraer a los expertos en el dominio de los detalles de implementación necesarios para definir los tipos de eventos y propiedades que describen cada uno de estos dominios. Asimismo, se presenta el editor gráfico creado para el modelado de dominios CEP y se detallan algunos de los aspectos más relevantes en cuanto a su implementación.

- El Capítulo 5 propone el DSML definido para el modelado de patrones de eventos, abstrayendo a los usuarios finales de los EPL requeridos para implementarlos, y se da a conocer tanto la sintaxis abstracta como la concreta de este DSML. Además, se describe el editor gráfico para el modelado de los patrones de eventos y su generación tanto a código EPL como al código de las acciones que deban ser ejecutadas en una SOA 2.0. Asimismo, se detallan algunos de los aspectos más relevantes en cuanto a su implementación, entre los que caben destacar la reconfiguración automática del editor dependiendo del dominio CEP que se desee utilizar, así como la personalización de la paleta del editor en tiempo de ejecución.

- El Capítulo 6 presenta la solución tecnológica propuesta para la integración de CEP con SOA 2.0, y su conexión con los editores gráficos implementados.

- En el Capítulo 7 se describen dos casos de estudios, el primero sobre la detección de situaciones relevantes en el ámbito de la domótica, y el segundo en el ámbito de la seguridad en redes. En cada caso de estudio, tras el modelado y la validación del dominio CEP, se modelan los patrones de eventos en los que, de forma gráfica e intuitiva, se especifican tanto las condiciones que deben cumplirse para detectar dichas situaciones como las acciones que deberían ser ejecutadas cuando acontezcan. Seguidamente, estos modelos son validados, almacenados y transformados al código que pueda ser desplegado automáticamente tanto en el motor CEP como en el ESB.

- En el Capítulo 8 se realiza una evaluación de los DSML definidos, los editores gráficos implementados, así como del enfoque dirigido por modelos propuesto en esta tesis doctoral.

- En el Capítulo 9 se recogen las contribuciones y las conclusiones que se han extraído de esta investigación, finalizando el capítulo con un conjunto de propuestas de líneas de investigación futuras relacionadas con esta tesis doctoral.
- En el Capítulo 10 se presenta la traducción al inglés del Capítulo 9.
- Además, se incluye un listado de los acrónimos utilizados, así como las referencias bibliográficas empleadas a lo largo de esta disertación.
- Finalmente, en el Anexo A se presenta el cuestionario proporcionado a los usuarios finales para la evaluación del editor gráfico de patrones de eventos.

Capítulo 2

Fundamentos y Estado del Arte

«Lo que sabemos es una gota, lo que ignoramos es un océano.»
(Isaac Newton)

Este capítulo se divide en dos partes bien diferenciadas. Por un lado, en la primera sección se definen los fundamentos en los que se basa esta tesis doctoral, así como se introducen y justifican las principales tecnologías usadas. Por otro lado, en la segunda sección se discuten los trabajos relacionados con las principales aportaciones derivadas de la realización de esta tesis.

2.1. Fundamentos

En esta sección se recopilan los conceptos fundamentales empleados en el desarrollo de esta tesis doctoral: desarrollo de software dirigido por modelos, arquitecturas orientadas a servicios, arquitecturas dirigidas por eventos, arquitecturas que combinan las orientadas a servicios con las dirigidas por eventos, así como el procesamiento de eventos complejos.

2.1.1. Desarrollo de Software Dirigido por Modelos

En esta sección se realiza una introducción a los conceptos principales del desarrollo de software dirigido por modelos y, a continuación, se detallan algunos de los *frameworks* y herramientas de Eclipse utilizadas en esta tesis doctoral para llevar a cabo este paradigma de desarrollo software.

Conceptos Fundamentales

En la actualidad, la comunidad de ingeniería del software se orienta más hacia el desarrollo de aplicaciones basadas en modelos, en vez de en tecnologías. El MDD se centra en los aspectos esenciales del sistema, retrasando a un paso posterior la elección de la tecnología más adecuada para su implementación [Ort07]. Las transformaciones entre modelos

y de modelo a código hacen realidad el desarrollo automático de sistemas; por lo que la definición del modelo y su transformación son las partes fundamentales del proceso. La gran reusabilidad y fiabilidad del código generado a través de este paradigma de construcción de software, así como el incremento de la productividad y un mantenimiento menos costoso, han conllevado a su aplicación en diversos ámbitos, tratando de satisfacer distintas necesidades tanto en el mundo empresarial como en la propia comunidad académica [GM13a].

En este contexto, un *modelo* es una representación simplificada de una determinada realidad con la intención de comprenderla mejor [Gén13]. Para lograr esta abstracción, los detalles irrelevantes de esta realidad deben ser apartados del modelo, que es independiente de cómo sea representado ya sea textual o gráficamente. A la representación gráfica de un modelo se le denomina *diagrama*.

Así pues, los modelos se expresan a través de *lenguajes de modelado* o DSML, cuya definición consta de tres partes bien diferenciadas: la *sintaxis abstracta* que está formada tanto por un *metamodelo* —un modelo que describe los conceptos del lenguaje y las relaciones entre estos— como por las *reglas de validación* que permiten determinar si un modelo está bien formado; la *sintaxis concreta* o notación del DSML —el conjunto de símbolos gráficos necesarios para dibujar los diagramas—; y las *transformaciones* entre modelos y de modelo a código para llevar a cabo la automatización del software. Conviene señalar que un mismo modelo, esto es, una instancia de un metamodelo, puede disponer de distintas notaciones gráficas. Fowler [Fow10] enumera cuáles son los beneficios de este tipo de lenguajes:

- *Mejora la productividad de desarrollo*: el problema a tratar para un dominio concreto se representa de una forma más abstracta, lo que facilita la especificación de qué tiene que hacer el sistema; además de evitar la aparición de errores y duplicación del código, ya que este se genera automáticamente desde modelos.

- *Mejora la comunicación con los expertos en el domino*: al tratarse de un lenguaje claro, conciso y cercano a los usuarios, fomenta las comunicaciones entre estos y el sistema software, quienes participan activamente en el modelado de lo que necesitan definir.

- *Facilita la adaptación ante los cambios*: el uso de modelos independientes de su implementación hace posible que puedan transformarse a código ejecutable en el entorno específico que se requiera usar en cada momento.

- *Especifica el qué, no el cómo*: este tipo de lenguajes ayuda al usuario en la tarea de especificar qué debe hacer el sistema, pero no el cómo debe llevarse a cabo.

Para la creación de un metamodelo se requiere el uso de un *lenguaje de metamodelado*, el cual dispone también de una sintaxis abstracta y una sintaxis concreta. En este tipo de lenguajes, la sintaxis abstracta se define con un *meta-metamodelo*, es decir, haciendo uso del propio lenguaje de metamodelado, mientras que la sintaxis concreta puede definirse bien mediante notaciones gráficas —los diagramas de clases de UML (*Unified Modeling*

Language) [OMG14c] son los más utilizados— o bien mediante notaciones textuales como, por ejemplo, Emfatic [Ecl12], una notación textual para expresar metamodelos Ecore.

Ecore es uno de los lenguajes de metamodelado más utilizados en la actualidad y forma parte de la arquitectura de metamodelado EMF (*Eclipse Modeling Framework*) [Ste08], definida por Eclipse. Concretamente, Ecore es una implementación del lenguaje EMOF (*Essential MOF*), un subconjunto del lenguaje MOF (*Meta-Object Facility*) [OMG14a] definido por la OMG (*Object Management Group*) [OMG14b] para la construcción de metamodelos.

EMF [Ecl14b] es un marco de trabajo de modelado que hace posible la creación de aplicaciones y herramientas basadas en un modelo de datos estructurado. A partir de una especificación de un modelo descrito en XMI (*XML Metadata Interchange*) [OMG14d], un formato estándar para el intercambio de metadatos independiente de plataforma, EMF proporciona herramientas y soporte en tiempo de ejecución para producir un conjunto de clases Java para el modelo, así como un editor básico. Los modelos pueden especificarse utilizando Java, documentos XML (*eXtensible Markup Language*), o bien mediante herramientas de modelado. Una vez creados estos modelos, podrán importarse en EMF.

Tanto los modelos como los metamodelos pueden ser almacenados a través de su serialización a XMI. Gracias a la serialización, estos modelos pueden compartirse entre múltiples herramientas de modelado.

Algunos autores [Atk01, GM13b, GP08, Kle03] han propuesto una *arquitectura de cuatro niveles* (véase la Figura 2.1) para explicar las relaciones existentes entre modelos y metamodelos, tal y como se detalla a continuación:

- *Nivel de datos (M0)*: representa los datos del mundo real. Estos datos son conformes a un determinado modelo.
- *Nivel de modelo (M1)*: caracteriza a los modelos que describen los datos del nivel M0. Todo modelo es conforme a un metamodelo.
- *Nivel de metamodelado (M2)*: caracteriza a metamodelos que describen los modelos del nivel M1. Todo metamodelo es conforme a un meta-metamodelo.
- *Nivel de meta-metamodelado (M3)*: caracteriza a los meta-metamodelos que describen los metamodelos del nivel M2. Un meta-metamodelo es conforme a sí mismo.

Con el fin de implementar editores que faciliten al usuario final la creación de modelos conformes a un metamodelo en particular, Eclipse proporciona varios marcos de trabajo o *frameworks*, entre los que caben destacar: GMF (*Eclipse Graphical Modeling Framework*) [Ecl14i, Gro09] para la construcción de editores gráficos de modelado y Xtext [Ecl14j] para editores textuales. Dada la complejidad que supone la implementación de editores basados en GMF, incrementada aún más por la ausencia de una buena documentación, el proyecto Epsilon [Ecl14c, Kol14] ofrece EuGENia [JR13], una herramienta cuya finalidad principal es precisamente reducir la curva de aprendizaje exigida para el desarrollo de estos editores gráficos GMF, automatizando parte de las tareas que con este *framework*

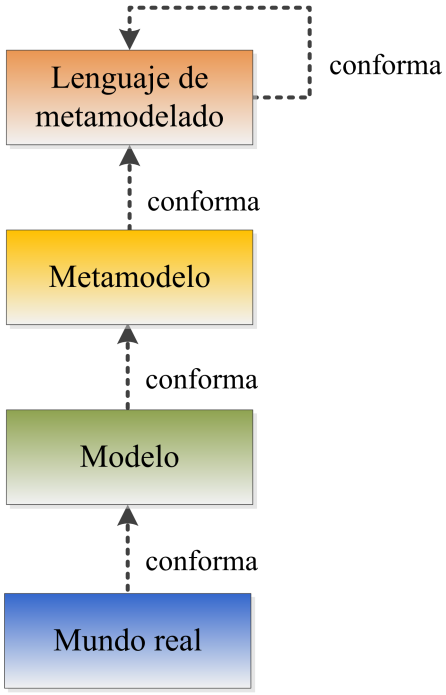

Figura 2.1: Arquitectura de cuatro niveles de modelado.

deberían realizarse manualmente. Por lo cual, reduce el esfuerzo necesario para producir, personalizar y mantener los modelos requeridos por GMF [Kol10].

Desarrollo de Software Dirigido por Modelos en Eclipse

Eclipse [Ecl14a] es una de las plataformas más utilizadas actualmente por la comunidad de ingeniería del software, debido a que es un proyecto de código abierto y, además, proporciona una gran cantidad de *frameworks* y herramientas de modelado que dan soporte tanto a la construcción como al despliegue y el mantenimiento del software. A continuación, se explican los dos marcos de trabajo que se han utilizado en esta tesis para la creación de los editores gráficos de modelado: GMF y Epsilon.

Graphical Modeling Framework GMF es un marco de trabajo que da soporte al desarrollo de editores gráficos de modelado, a partir de metamodelos EMF y capas gráficas basadas en GEF [Ecl14h]. Mientras que EMF proporciona la infraestructura de modelado necesaria para implementar estos editores, el marco de trabajo GEF ofrece los mecanismos necesarios para la creación de los diagramas.

Este marco de trabajo requiere la definición de unos modelos concretos, a partir de los cuales GMF generará el código que implemente un editor gráfico, obtenido como un *plugin* de Eclipse. Los modelos requeridos son los siguientes:

- *Modelo de dominio*: el metamodelo Ecore (*.ecore*) que describe la sintaxis abstracta del DSML.

- *Modelo de definición de los gráficos*: el modelo (*.gmfgraph*) que especifica los elementos gráficos que se mostrarán en los diagramas.
- *Modelo de definición de la paleta*: el modelo (*.gmftool*) que define la paleta de herramientas del editor.
- *Modelo de correspondencias*: el modelo (*.gmfmap*) que establece la correspondencia de cada elemento del metamodelo con su representación gráfica y la herramienta de la paleta que permitirá añadir el elemento en cuestión al diagrama.

Téngase en cuenta que GMF es capaz de generar automáticamente los modelos *.gmfgraph*, *.gmftool* y *.gmfmap* a partir de un metamodelo Ecore. No obstante, estos modelos tendrán que ser modificados en profundidad por el programador para crear la versión definitiva del editor. Por lo cual, la creación de editores gráficos con GMF no es una tarea trivial, en absoluto.

Cabe destacar que, al utilizar un editor GMF para la definición de un modelo, se crearán dos ficheros: un modelo EMF serializado en formato XMI y un diagrama, que almacenará toda la información relativa a la parte visual del modelo como, por ejemplo, la posición, el color y el tamaño de los elementos gráficos. Gracias a la especificación XMI podrá compartirse dicho modelo entre diferentes herramientas de modelado.

Epsilon Epsilon [Kol14] es una familia de lenguajes de propósito específico que pueden trabajar directamente sobre modelos y que ofrecen soporte para las principales tareas relacionadas con MDD, entre las que caben destacar, la validación de modelos —mediante el uso de EVL (*Epsilon Validation Language*) [Ecl14g]—, las transformaciones de modelo a modelo —mediante el uso de ETL (*Epsilon Transformation Language*) [Ecl14f]— y las transformaciones de modelo a código —mediante el uso de EGL (*Epsilon Generation Language*) [Ecl14d]. El núcleo de Epsilon es EOL (*Epsilon Object Language*) [Ecl14e], un lenguaje imperativo orientado a modelos, a partir del cuál se construye toda la familia de lenguajes Epsilon.

De todos los lenguajes proporcionados por Epsilon, en este trabajo se han utilizado los siguientes lenguajes:

- EOL: un lenguaje de programación imperativo para la creación, la consulta y la modificación de modelos EMF. Este lenguaje proporciona todas las características imperativas del lenguaje JavaScript —bucles *for* y *while*, variables, estructuras condicionales *if*, entre otros— junto con otras características propias del lenguaje OCL (*Object Constraint Language*) [OMG14b] como, por ejemplo, las funciones de consulta sobre colecciones de datos.
- EVL: un lenguaje construido sobre EOL. Las restricciones EVL son similares a las restricciones OCL. Sin embargo, EVL presenta algunas características adicionales, no disponibles en OCL: dependencias entre restricciones, personalización de los mensajes de error que deben notificarse a los usuarios, así como la especificación de reparaciones

que los usuarios pueden invocar con el fin de solucionar las inconsistencias presentes en los modelos.

- EGL: un lenguaje basado en plantillas para la transformación de modelo a texto, así como la generación de documentación y otros artefactos textuales a partir de modelos. Entre sus características principales se encuentran las siguientes: desacopla el contenido del destino —haciendo posible la generación de texto a ficheros, al portapapeles, etc.—, permite que unas plantillas sean invocadas por otras y proporciona regiones protegidas para mezclar el código generado automáticamente con el código escrito manualmente.

Asimismo, Epsilon proporciona la herramienta EuGENia, que tiene por finalidad facilitar el desarrollo de editores gráficos basados en GMF, ya que este *framework* de modelado gráfico de Eclipse, como se ha mencionado previamente, es bastante complejo y no presenta una buena documentación teniendo, por tanto, una curva de aprendizaje elevada.

Una de las grandes ventajas de utilizar EuGENia es que, a partir de un metamodelo especificado en formato Ecore o Emfatic y enriquecido con anotaciones específicas de GMF, se pueden generar automáticamente los tres modelos requeridos por GMF para implementar un editor gráfico: *gmfgraph*, *gmftool* y *gmfmap*.

Otra de las ventajas que aporta EuGENia es que estos modelos GMF pueden ser personalizados automáticamente mediante transformaciones de pulido (*polishing transformations*) escritas en EOL. Así pues, en el caso de que fuese necesario realizar alguna modificación en el metamodelo de partida, esto no supondrá ningún problema puesto que se regenerarán automáticamente dichos modelos GMF y, a continuación, se aplicará la personalización implementada específicamente para ellos; lo que conlleva un ahorro de tiempo muy significativo. Esta ha sido la principal razón por la que se ha decidido utilizar Epsilon en esta tesis doctoral frente a otras alternativas existentes en la actualidad.

2.1.2. Arquitecturas Orientadas a Servicios

Una arquitectura orientada a servicios es una forma lógica de diseñar un sistema software con el propósito de ofrecer servicios bien a las aplicaciones de usuario final o bien a otros servicios distribuidos a lo largo de una red; todo ello, haciendo uso de interfaces públicas y visibles [Pap07a].

Chandy y Schulte determinan que cualquier aplicación que implemente SOA debe cumplir los siguientes principios [Cha10]:

1. *Aplicación modular*: la aplicación debe ser modular, para que los componentes software (agentes) pueden añadirse, modificarse o eliminarse individualmente.

2. *Componentes distribuibles*: los componentes deben ser capaces de ejecutarse en ordenadores diferentes y comunicarse con otros mediante el envío de mensajes a través de la red y en tiempo de ejecución.

3. *Interfaces de componente visibles*: estas interfaces deben estar accesibles a otros desarrolladores de aplicaciones. Además, estas interfaces deben definirse y documentarse claramente en los metadatos. Estos describen los mensajes de entrada y de salida de cada componente, así como la suficiente información como para que los desarrolladores puedan encontrar y usar el componente como parte de una nueva aplicación.

4. *Componentes reemplazables*: un componente que proporcione un servicio podrá reemplazarse por otro componente que ofrezca el mismo servicio, siempre y cuando ambos compartan la misma interfaz. Esto es posible debido a que el diseño de la interfaz, es decir, lo que hace, está separado de la implementación del servicio interno, esto es, cómo está hecho.

5. *Componentes compartibles y reutilizables*: el mismo código y los mismos datos están disponibles para los usuarios de cualquier aplicación que comparta el mismo componente software.

La combinación de los cuatro primeros principios implica que los componentes SOA están *débilmente acoplados*, lo cual facilita la consecución del quinto principio.

Por consiguiente, este tipo de arquitecturas proporciona un entorno débilmente acoplado con múltiples funcionalidades, mejorando notablemente el mantenimiento y la evolución de los sistemas de la empresa.

2.1.3. Arquitecturas Dirigidas por Eventos

Una arquitectura dirigida por eventos es un patrón de arquitectura software en el que algunos de los componentes de un sistema son dirigidos por eventos y están mínimamente acoplados [Cha10]. Un *evento* se define como algo que ocurre o que se espera que ocurra [Eve10]. Por otro lado, Chandy y Schulte definen el concepto *dirigido por eventos* como el comportamiento de cualquier entidad que reacciona cuando reconoce un evento, y el concepto *mínimamente acoplado* como la existencia de una relación unidireccional entre un productor de eventos y un consumidor de eventos en la cual el productor es quien envía los eventos al consumidor.

Asimismo, estos autores determinan que cualquier aplicación de negocio que implemente EDA debe cumplir los siguientes principios [Cha10]:

1. *Notificación de eventos*: deberá informarse en cuanto acontezca un nuevo evento en el sistema.

2. *Envío de notificaciones*: el productor de eventos será el que decida cuándo enviar las notificaciones de los eventos al consumidor de eventos.

3. *Respuesta inmediata*: el consumidor de eventos reaccionará, llevando a cabo alguna acción, tan pronto como reconozca un evento.

4. *Comunicación unidireccional*: el productor de eventos enviará las notificaciones de los eventos al consumidor de eventos, sin esperar ningún tipo de respuesta por parte del consumidor.

5. *Notificaciones libre de peticiones*: las notificaciones de eventos son meros "informes" en los que bajo ningún concepto se detallará ninguna de las acciones que deban ser ejecutadas por el consumidor de eventos, tras su recepción.

Los tres primeros principios aluden al concepto de *dirigido por eventos*, mientras que los dos últimos engloban el significado de *mínimamente acoplado*. Conviene mencionar que podría darse el caso en el que un sistema fuese dirigido por eventos, a pesar de no usar una EDA. Por ejemplo, considérese un productor de eventos que envíe una notificación al consumidor de eventos en la que se especifique la acción concreta que deba ser ejecutada por dicho consumidor.

Estos cinco principios de EDA pueden implementarse mediante un patrón de comunicación *publicación/suscripción* [Fai06, Müh06, Tay08]. Con este tipo de comunicaciones asíncronas, un mensaje puede entregarse a cero, uno o varios consumidores sin que el productor tenga que enviarlo varias veces. Por otra parte, una suscripción determina el tipo de mensajes que deben ser enviados a cada consumidor. Es importante destacar que el mecanismo de publicación/suscripción es mucho más flexible que otros tipos de mecanismos de mensajería porque las instrucciones de entrega de los mensajes no se definen estáticamente y, además, las suscripciones pueden llevarse a cabo en tiempo de ejecución. Otra de sus principales ventajas es que nuevos productores o consumidores de eventos pueden ser añadidos en cualquier momento; del mismo modo, podrán eliminarse si así se requiere.

2.1.4. Arquitecturas Orientadas a Servicios y Dirigidas por Eventos

Una arquitectura orientada a servicios y dirigida por eventos, también conocida como ED-SOA o SOA 2.0, es una evolución de una SOA tradicional en la que las comunicaciones entre usuarios y servicios se lleva a cabo a través de eventos, en lugar de mediante llamadas a procedimientos remotos [Luc12].

Cualquier aplicación que implemente SOA 2.0 deberá cumplir tanto los cinco principios de SOA (véase la Sección 2.1.2) como los cinco principios de EDA (véase la Sección 2.1.3).

Esto pone de manifiesto que EDA puede complementar a SOA, por lo que no son alternativas excluyentes entre sí, sino compatibles y complementarias. Para lograr esta integración, se requiere una capa de abstracción de software altamente distribuida e integradora [Pap06, Pap07b]. Estas funcionalidades son ofrecidas por un ESB, un *middleware* que posibilita la interoperabilidad entre protocolos de comunicación diferentes, pudiéndose usar como una plataforma de integración donde las aplicaciones son expuestas como servicios [Dav09].

Según Rademakers y Dirksen [Rad09], las funcionalidades principales de un ESB son las siguientes: transparencia de localización, conversión de protocolos de transporte, transfor-

mación de mensajes, encaminamiento de mensajes, enriquecimiento de mensajes, seguridad, así como gestión y monitorización.

En cuanto a las soluciones ESB actuales, se ha seleccionado Mule [Dos14, Mul14], de la compañía MuleSoft, como el candidato ideal para dicha integración. La elección se ha realizado, en primer lugar, por ofrecer un editor gráfico que pone al alcance de cualquier programador no experto en ESB la definición gráfica e intuitiva de los flujos que implementan la aplicación Mule en cuestión. Por otro lado, un informe realizado recientemente por Forrester [Rie14] confirma que MuleSoft es una de las soluciones más usadas en la actualidad y mejor valoradas en cuanto a su capacidad para integrarse con plataformas *cloud* [Arm10, Sos11], así como con múltiples herramientas y escenarios.

A pesar de todas las ventajas que proporciona la combinación de EDA y SOA, este tipo de arquitectura SOA 2.0 no es adecuada para procesar, analizar y correlacionar una ingente cantidad de información, en forma de eventos, con el fin de detectar en tiempo real situaciones críticas o excepcionales para un dominio en particular. Téngase en cuenta que a partir de los datos que fluyan por una SOA 2.0, una empresa podría estar interesada en obtener cierta información que suponga un valor añadido frente a sus competidores tan pronto como sea posible.

Así pues, para cubrir estas necesidades de los expertos en el negocio, no cubiertas por SOA 2.0, se propone la integración de la tecnología CEP con SOA 2.0.

2.1.5. Procesamiento de Eventos Complejos

A continuación, se define en qué consiste la tecnología CEP, cuáles son sus principales ventajas, así como algunos de los dominios más relevantes donde se puede aplicar. Seguidamente, se analizan los tipos de lenguajes específicos que existen para hacer uso de esta tecnología.

Conceptos Fundamentales

Como se ha comentado previamente, CEP es una tecnología emergente que permite capturar, analizar y correlacionar una ingente cantidad de eventos heterogéneos con el fin de detectar situaciones críticas o relevantes en tiempo real. Un *evento* es algo que ocurre o que se espera que ocurra [Eve10], también puede ser algo que se espera que ocurra pero esto no llega a suceder; en el contexto de un sistema de información, podría definirse como un mensaje que contiene información sobre lo que justamente está ocurriendo en un momento determinado [Luc12]. Por otra parte, una *situación* es una ocurrencia de un evento o una sucesión de eventos que requiere alguna reacción inmediata [Etz10].

Esta tecnología se basa en el filtrado de eventos irrelevantes y en el reconocimiento de los eventos que sí son relevantes para un dominio en particular. Para ello, se utilizan unos *patrones de eventos*, esto es, unas plantillas en las que se especifican cuáles son las condiciones que deben cumplirse para detectar dichas situaciones de interés, así como las acciones que deberán ser ejecutadas tras su detección. A estas situaciones de una mayor

complejidad semántica se les denominan *eventos complejos*, por tanto, un evento complejo es una abstracción de otros eventos simples o complejos que contiene la información necesaria para describir la situación recién acontecida [Luc12].

La característica principal de estos eventos complejos procesados mediante tecnología CEP es que pueden ser identificados e informados en tiempo real, reduciendo la latencia en la toma de decisiones, a diferencia del software tradicional de análisis de eventos que no funciona en tiempo real —normalmente procesa eventos almacenados en bases de datos. Además, los eventos complejos pueden ser estructurados mediante una *jerarquía de eventos*, donde cada evento de un nivel superior se habrá inferido a partir de uno o más eventos de un nivel inferior. Gracias a esta jerarquerización, se logrará reducir el número de eventos que deberá procesarse según el nivel de abstracción requerido por el usuario y/o sistema en cuestión en cada momento, entregándose de esta forma la información correcta al usuario y/o sistema adecuado.

Para llevar a la práctica este tipo de procesamiento de eventos en los sistemas de información, es necesario hacer uso de un software específico conocido como *motor CEP*. En la actualidad existen varios motores CEP desarrollados por distintas empresas y grupos de investigación. Cada motor proporciona al programador un lenguaje concreto para que implemente los patrones de eventos que desee detectar en tiempo real; a este tipo de lenguaje se le denomina *lenguaje de procesamiento de eventos* o EPL; en la siguiente subsección se proporcionan detalles adicionales sobre los motores CEP más relevantes, y sus respectivos EPL.

Como muestra la Figura 2.2, este tipo de procesamiento de eventos se efectúa en tres etapas [Cha10]:

1. *Captura de eventos*: consiste en la recepción de los eventos que se analizarán con la tecnología CEP.

2. *Análisis*: a partir de los patrones de eventos definidos previamente en el motor CEP, este se encargará de procesar y correlacionar toda la información, en forma de eventos, con el fin de detectar situaciones críticas o relevantes en tiempo real.

3. *Respuesta*: tras la detección de una determina situación, se actuará en consecuencia notificándoselo al sistema, software, servicio o dispositivo en cuestión.

Entre las ventajas más relevantes ofrecidas por esta tecnología se encuentran las siguientes [Cha10, Dag10]:

- *Mejora de la calidad en las decisiones*: los ordenadores son capaces de gestionar muchísima más información por segundo y de contemplar muchos más factores a la hora de tomar una decisión frente a las capacidades de que disponen los seres humanos.

- *Respuesta veloz*: el uso de un motor CEP que permite realizar automáticamente tanto el análisis de los eventos como la realización de la toma de decisiones, sin que sea necesario la intervención de ningún usuario, conllevará a una rápida respuesta.

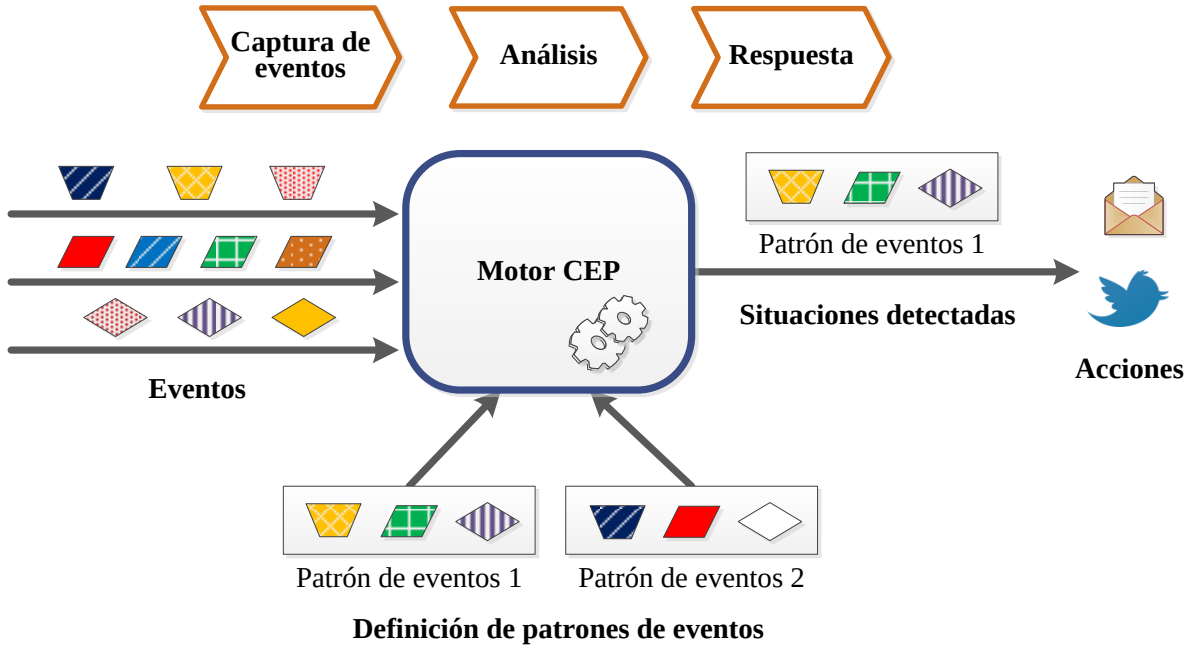

Figura 2.2: Etapas implicadas en el procesamiento de eventos complejos.

- *Prevención de sobrecarga de información*: debido a la naturaleza inherente de los sistemas CEP, se reduce enormemente la cantidad de información que debe ser notificada a las personas, puesto que normalmente solo se mostrarán a los usuarios finales aquellas situaciones críticas o relevantes (eventos complejos) en las que estén interesados; estas situaciones se habrán detectado a través de los patrones de eventos y tras un análisis exhaustivo de toda la información recibida.

- *Reducción del esfuerzo humano*: puesto que se utilizan sistemas informáticos, se reduce el esfuerzo y el trabajo necesarios por parte de las personas para analizar toda la información.

Por consiguiente, puede aplicarse CEP a una gran cantidad de dominios en los que la detección de situaciones críticas o relevantes sea una necesidad. Algunos de los dominios que pueden beneficiarse de esta tecnología son los siguientes: detección de fraude y seguridad informática [DC12, Gad12a, Gad12b, Gad13], domótica [BP14a, Rom11], salud [BP11, BP14b, Yua09], gestión del tráfico marítimo [BP12], energía y fabricación [Dun11], servicios basados en localización [Vik13], sistemas y operaciones financieros [Uhm11] e inteligencia de negocios operacional [Cha11].

Lenguajes de Procesamiento de Eventos

Gracias al uso de patrones de eventos en sistemas CEP es posible la detección de situaciones de interés sobre un dominio concreto en tiempo real. Estos patrones de eventos

se definen utilizando unos lenguajes de programación desarrollados para tal fin, denominados EPL, como se ha comentado previamente. Estos lenguajes pueden clasificarse en tres categorías [Etz10]:

- *EPL orientados a flujos*: estos lenguajes están basados en SQL (*Structured Query Language*) y el álgebra relacional, pero extendidos para que puedan manejar ventanas temporales de datos. A diferencia de los modelos relacionales convencionales en los que una consulta se ejecuta sobre una tabla de datos, este modelo de consulta continua y en tiempo de ejecución deberá ejecutarse sobre un conjunto limitado de eventos (normalmente agrupados mediante ventanas temporales). Algunos de estos EPL orientados a flujos son los siguientes: EPL de Esper [Esp14], EPL y CQL (*Continuous Query Language*) de Oracle [Ora14], StreamSQL [Str14] y CCL (*Continuous Computation Language*) [Syb14].

- *EPL orientados a reglas*: este tipo de EPL se clasifica, a su vez, en tres subtipos:

  - *Reglas de producción*: estas reglas son del tipo *si condición entonces acción*, esto es, cuando la condición se satisface, entonces se ejecutará la acción. El EPL del motor Drools Fusion [JBo14] es uno de los más conocidos entre los lenguajes de esta categoría.
  - *Reglas activas*: también denominadas reglas de *evento-condición-acción* o ECA (*Event-Condition-Action*) están inspiradas en las bases de datos activas que siguen el siguiente funcionamiento: cuando un evento ocurre, se evalúan las condiciones y, si se satisfacen, entonces se lanza la acción correspondiente. Entre los EPL de reglas activas se encuentra, por ejemplo, el proporcionado por IBM Operational Decision Management [IBM14].
  - *Reglas de programación lógica*: estos lenguajes están basados en aserciones lógicas y en bases de datos deductivas, que permiten hacer deducciones a través de inferencias, a partir de reglas y un conjunto de hechos concreto. Como ejemplo, destacar ETALIS (*Event TrAnsaction Logic Inference System*) [Ani14], un sistema CEP implementado en Prolog que ofrece dos lenguajes basados en reglas: ELE (*ETALIS Language for Events*) y EP-SPARQL (*Event Processing SPARQL*).

- *EPL imperativos*: lenguajes que permiten definir, mediante programación imperativa, el conjunto de operadores que debe aplicarse sobre los eventos, en el que cada operador especifica una transformación sobre los eventos recibidos. El EPL de Apama [AG14] es el lenguaje más representativo de los EPL imperativos.

Puede encontrarse información más exhaustiva sobre estos tipos de EPL en el estudio realizado por Cugola y Margara [Cug12].

Es importante destacar que en esta tesis doctoral se ha optado por utilizar el EPL de Esper, cuya sintaxis se aproxima bastante a la del lenguaje SQL, ampliamente conocido

a nivel mundial, por lo tanto, la curva de aprendizaje no es elevada para el programador. Además, este lenguaje soporta de forma nativa varios tipos de formato de eventos: objetos Java/.NET, documentos XML y *maps* de Java. Este último tipo de formato es muy ventajoso con respecto a otros, ya que permite definir tipos de eventos y propiedades de eventos anidadas en tiempo de ejecución, sin necesidad de crear ningún fichero adicional en tiempo de diseño. Asimismo, este lenguaje se ejecuta en Esper, un motor CEP escrito en Java y de código abierto, y uno de los más utilizados en la actualidad, capaz de procesar en torno a 500.000 eventos/s en una estación de trabajo, y entre 70.000 y 200.000 eventos/s en un portátil, según confirma EsperTech, la empresa desarrolladora.

2.2. Estado del Arte

En esta sección se presentan los trabajos relacionados con los diversos aspectos que se abordan en esta tesis doctoral. En primer lugar, se describen los trabajos que proponen enfoques para representar los eventos y los patrones de eventos para CEP. En segundo lugar, se enumeran los editores gráficos para la definición y la generación de código de patrones de eventos más representativos que se han desarrollado tanto en el mundo empresarial como en el académico e investigador. Finalmente, se citan las propuestas existentes para la integración de CEP con SOA, EDA y SOA 2.0.

2.2.1. Enfoques para el Procesamiento de Eventos Complejos

Durante los últimos años, se han propuesto distintos enfoques con el fin de incorporar el conocimiento del dominio en CEP, así como tratar de acercar esta tecnología a los expertos en el dominio para ayudarles en el proceso de la toma de decisiones. Algunos de estos enfoques persiguen la integración del conocimiento semántico en CEP haciendo uso de lenguajes ontológicos para modelar los dominios. Otros enfoques emplean técnicas de MDD con la finalidad de definir los dominios como modelos, que posteriormente serán transformados a código específico del sistema CEP que se requiera usar.

Tras una extensa revisión bibliográfica sobre dichos tipos de enfoques existentes para CEP, en primer lugar, se detallan los trabajos relacionados sobre enfoques que utilizan ontologías y, en segundo lugar, aquellos trabajos que proponen enfoques dirigidos por modelos.

Enfoques Ontológicos

Sen *et al.* [Sen10a, Sen10b] han propuesto un modelo semántico para la representación de eventos y patrones de eventos basado en RDFS (*Resource Description Framework Schema*) [W3C14b], un lenguaje de representación del conocimiento que proporciona los elementos necesarios para la descripción de ontologías. Concretamente, la ontología que han definido contiene un conjunto de conceptos (*Event*, *EventOperator*, *EventSource* y *EventType*) y un conjunto de propiedades (*hasStartTime*, *hasEndTime*, *hasEventName*,

hasEventId, *hasEventSource*, *hasEventType* y *composedOf*). Los conceptos *EventType* y *EventSource* permiten clasificar los eventos con características similares en cuanto al tipo y la fuente de donde provengan. Cada evento tiene un identificador único (*hasEventId*) y un nombre (*hasEventName*). Además, mientras un evento simple es indivisible y ocurre en un momento concreto (*hasStartTime*), un evento complejo puede ocurrir durante un período de tiempo, comprendido entre la fecha de inicio (*hasStartTime*) y la fecha de fin (*hasEndTime*). Cada evento complejo, al estar compuesto de otros eventos, tendrá adicionalmente una propiedad denominada *composedOf*. Conviene mencionar que los operadores de eventos considerados en esta ontología son los mismos que los propuestos por Chakravarthy y Mishra [Cha94], esto es, operadores de eventos de agregación (COUNT), de ventanas de datos (WITHIN), lógicos (AND y OR) y temporales (SEQ). En este modelo semántico, un patrón de eventos se describe con un conjunto de eventos junto con un conjunto de operadores de eventos.

Este modelo propuesto por Sen *et al.* para la representación de eventos y patrones de eventos utilizando ontologías se ha utilizado con algunas modificaciones en el proyecto europeo ALERT (*Active support and reaL-time coordination based on Event pRocessing in FLOSS developmenT*) [ALE13] del VII Programa Marco de Investigación y Desarrollo o FP7 (*Framework Programme 7*). Concretamente, se ha añadido a dicho modelo un nuevo concepto denominado *interaction pattern*, que extiende la representación del evento con un conjunto de propiedades que facilitan la definición de un patrón de eventos: el identificador, el nombre y la fecha de creación del patrón. Al igual que en el modelo anterior, se utilizan operadores de eventos de agregación, de ventanas de datos, lógicos y temporales.

Stühmer *et al.* [Stü09] también presentan una ontología RDFS para eventos, donde un evento puede representarse como un evento simple o un evento complejo. Todos los eventos tienen un tipo, fecha y hora de comienzo, fecha y hora de fin, y opcionalmente el contenido del mensaje. A diferencia de otras propuestas, en esta se especializa el evento complejo con los tipos de operadores de eventos que, en cada caso, se requieran para detectarlo, tales como *AndEvent*, *OrEvent*, *NotEvent.*

Paschke *et al.* [Pas12] han creado el modelo Reaction RuleML, que define los siguientes conceptos en una estructura ontológica: evento, situación, espacio, tiempo, acción y agente. Estos conceptos pueden relacionarse entre sí y, además, especializarse con ontologías de dominio existentes —vocabularios relacionados con un dominio específico— y ontologías para tareas y actividades genéricas como, por ejemplo, el procesamiento de situaciones de interés. Cada situación de interés estará compuesta por sus propiedades (*hasProperties*) y una descripción (*hasContent*), además de pertenecer a una de las siguientes categorías: heterogéneas —aquellas influenciadas por cambios dinámicos, tiempo o frecuencia— y homogéneas —descritas por situaciones estables, iterativas o habituales.

Previamente a la creación del modelo Reaction RuleML, Paschke publicó en solitario el modelo de lenguaje de patrón CEP [Pas09]. Este lenguaje se ha implementado como un lenguaje de descripción de patrones formado por una combinación del lenguaje XML, una descripción en lenguaje natural (en lengua inglesa) y un lenguaje ontológico implementado como un modelo de vocabulario OWL (*Web Ontology Language*) [W3C14a].

Yao *et al.* [Yao11] también han propuesto una ontología CEP. Esta está formada por

eventos que se clasifican en eventos simples y eventos complejos. Cada evento tiene un tipo y una serie de atributos. Los eventos simples se clasifican, a su vez, en eventos RFID (*Radio Frequency IDentification*) y en eventos no RFID. Por otro lado, los eventos complejos se componen de operadores de eventos, que pueden ser de dos tipos: lógicos y temporales. Asimismo, estos eventos complejos se asocian con los patrones de eventos que se encargarán de lanzar las acciones pertinentes. Esta ontología se ha aplicado al ámbito de la salud, concretamente, se ha monitorizado toda la información de un hospital, donde se han instalado dispositivos RFID para tal fin.

Todos estos enfoques ontológicos citados difieren en gran medida del enfoque dirigido por modelos propuesto en esta tesis doctoral (véase el Capítulo 3). Por un lado, el número de operadores incorporado en cada uno de los modelos mencionados es inferior al número de operadores que se han incluido en los DSML gráficos que se proponen en esta tesis tanto para la definición de dominios CEP (véase el Capítulo 4) como para la definición de patrones de eventos (véase el Capítulo 5). Asimismo, el número de operandos de dimensión temporal y de las ventanas de datos es inferior en dichos modelos. Esto implica una limitación en la expresividad de los patrones más complejos que se requiera modelar.

Por otro lado, la forma en la que se definen algunos modelos difiere bastante de cómo se han definido los metamodelos en la tesis. Por ejemplo, el modelo de Stühmer representa los operadores como clases especializadas de la clase *ComplexEvent*, en lugar de crear una clase operador con sus operadores como subclases de esta; tampoco se determina la aridad de dichos operadores. Todo ello, dificulta la escalabilidad y la comprensión de los patrones definidos.

Enfoques Dirigidos por Modelos

Una vez analizados los trabajos que hacen uso de lenguajes ontológicos, seguidamente se presentan los enfoques dirigidos por modelos, clasificados según proporcionen un lenguaje textual para definir los patrones o, por el contrario, un lenguaje gráfico.

En cuanto a los enfoques que proporcionan una representación textual de los patrones de eventos se encuentran los metamodelos de Zang *et al.* y Bruns *et al.*, que se describen a continuación.

Zang *et al.* [Zan08] han definido un metamodelo en el que los eventos se interrelacionan con operadores de eventos, procesos y un contexto. Estos eventos pueden provenir de distintas fuentes, tales como servicios, bases de datos, dispositivos RFID y actividades de procesos. Del mismo modo que en propuestas anteriores, los tipos de eventos, junto con las propiedades, se clasifican en simples y complejos, que pueden asociarse a través de relaciones de causalidad. En este metamodelo se definen los siguientes tipos de operadores: lógicos, temporales, causales y de RFID. Estos hacen posible la combinación de varios eventos con el fin de crear situaciones (eventos complejos). Asimismo, estos eventos pueden formar parte de un contexto, gracias al cual será posible crear una jerarquía de eventos.

Bruns *et al.* [Bru14] han definido un DSML textual, denominado DS-EPL (*Domain-Specific Event Processing Language*), para la definición de patrones de eventos sobre un dominio específico, en concreto, un lenguaje para modelar patrones de eventos del dominio

máquina-a-máquina que permitirán detectar situaciones de interés entre máquinas y dispositivos remotos, que intercambian información haciendo uso de redes tanto alámbricas como inhalámbricas. Además, los autores evalúan este DSML aplicándolo a un caso de estudio sobre una planta de energía solar. Este enfoque exige la creación de un DSML por cada dominio donde deba aplicarse CEP, así como la especificación predeterminada de un conjunto de tipos de eventos para el dominio en cuestión. A diferencia de la mayoría de los enfoques analizados, en este se proponen operadores que permiten especificar condiciones cualitativas como, por ejemplo, se ha incrementado (operador *increased*) o disminuido (operador *decreased*) el valor de cierta propiedad de un evento. Asimismo, se ha construido un editor textual con el marco de trabajo Xtext como soporte para el modelado de los patrones textuales.

A pesar de que los metamodelos de estos lenguajes tienen en cuenta el dominio al que pertenece un evento concreto, ninguno ofrece la posibilidad de describir un dominio CEP compuesto de un conjunto de tipos de eventos, junto con su descripción y fecha de creación, lo cual facilitaría que los dominios pudiesen compartirse entre distintas herramientas de modelado que, incluso, podrían pertenecer a distintos usuarios. Esta posibilidad sí se ha contemplado en el enfoque de esta tesis.

Aunque los autores de ambos enfoques argumentan la utilidad y la expresividad de sus lenguajes textuales, haciendo mención expresa a que facilitan el modelado de patrones de eventos a los usuarios que son expertos en el dominio, pero no expertos en CEP, estos enfoques presentan un grado de usabilidad y funcionalidad inferior al propuesto en esta tesis. Por un lado, un lenguaje gráfico bien definido normalmente será más amigable e intuitivo para un usuario no tecnólogo que un lenguaje textual, ya que los usuarios finales suelen tener cierto rechazo al uso de aplicaciones y herramientas que no dispongan de una interfaz gráfica.

Además de tratarse de un lenguaje textual, el lenguaje DS-EPL presenta otras limitaciones con respecto a los DSML propuestos en esta tesis. En primer lugar, aunque sus autores consideran una ventaja que el DSML sea específico de un dominio concreto —máquina-a-máquina, en este caso—, esto requiere crear un DSML por cada dominio donde se necesite aplicar CEP, lo que implica una mayor carga de trabajo y esfuerzo adicional por cada nuevo DSML a implementar. Para solventar este problema, en esta tesis se propone un DSML de patrones de eventos independiente del dominio donde se necesite aplicar dicha tecnología, pero con la capacidad de personalizarse a cualquier dominio CEP, a partir de los tipos de eventos modelados para el nuevo dominio a incorporar. En segundo lugar, el lenguaje DS-EPL presenta un conjunto predefinido de tipos de eventos que puede extenderse para definir otros nuevos; sin embargo, no se contempla la posibilidad de crear otros tipos de eventos que no dependan de los ya proporcionados inicialmente. En tercer lugar, los autores de dicho lenguaje proponen operadores que representan relaciones de dominio cualitativas; no obstante, un usuario final también podría necesitar definir condiciones concretas que determinen qué se entiende cuantitativamente por un aumento del valor de una propiedad, en lugar de utilizar un valor cualitativo como *aumentado* (*increased*). Finalmente, tampoco consideran la inclusión en el lenguaje de posibles acciones a llevar a cabo cuando se creen los eventos complejos.

Conviene destacar que existen otros enfoques dirigidos por modelos para CEP que proporcionan lenguajes textuales. Sin embargo, estos enfoques quedan fuera del alcance del estado del arte de esta tesis doctoral, puesto que se han implementado con el propósito de definir patrones de eventos restringidos a un dominio en particular. Por ejemplo, Mulo *et al.* [Mul13] han propuesto un enfoque para la monitorización del grado de cumplimiento en SOA dirigidas por procesos. Concretamente, definen un DSML textual para la especificación de directivas del grado de cumplimiento de procesos, así como la transformación de estas directivas en sus correspondientes patrones de procesos de negocio.

En cuanto a los enfoques que proporcionan una representación gráfica de los patrones de eventos, se encuentran los metamodelos de Obweger *et al.*, y Etzion y von Halle, detallados a continuación.

Obweger *et al.* [Obw10, Obw11] han propuesto un modelo de eventos, un modelo de correlación y un modelo de reglas, entre otros. El modelo de eventos permite definir los tipos de eventos que se manipulan en el sistema, que pueden clasificarse en tres tipos: tipos de valores simples —numéricos, cadena de caracteres y otros tipos de eventos—, colecciones de datos y diccionarios de datos. Por otro lado, el modelo de correlación ofrece la posibilidad de establecer relaciones entre los eventos. Finalmente, a través del modelo de reglas se describen las situaciones que se desean detectar (eventos complejos). Cabe destacar que la definición de un patrón viene dada fundamentalmente por un conjunto de componentes, un conjunto de relaciones de precondición, un conjunto de entradas y un conjunto de salidas. Estos componentes permiten establecer las condiciones que deben cumplirse para la detección de las situaciones de interés, especificar intervalos temporales, así como indicar si un patrón depende de otro patrón ya definido. Además, estos componentes pueden relacionarse mediante enlaces que unan los puertos de salida de un componente —acciones a llevar a cabo tras la detección de la situación definida— con los puertos de entrada de otro componente —las precondiciones que deben cumplirse para analizar el contenido del componente en cuestión. De este modo, los patrones se modelan como grafos de decisión visuales.

Etzion y von Halle [Etz13] han presentado un enfoque dirigido por modelos para la definición de patrones de eventos denominado *The Event Model*. Este enfoque se fundamenta en los conceptos descritos por el autor principal en [Etz10]. Según los autores, las características de este enfoque son las siguientes: proporciona una estructura para llevar a cabo un modelado riguroso de la realidad, los patrones se representan haciendo uso de tablas y estos se pueden transformar automáticamente al código de un EPL específico, y los modelos son independientes de la implementación.

Cabe destacar que de los dos últimos enfoques dirigidos por modelos mencionados, el que se asemeja más al propuesto en esta tesis doctoral es el enfoque descrito por Etzion y von Halle. No obstante, una de las diferencias más destacables entre ambos es el modo en que los expertos en el negocio pueden describir sus situaciones de interés: mientras que estos autores han optado por el uso de plantillas en formato tabla con textos para definir los patrones de eventos, en el enfoque de la tesis se ha elegido un formato más gráfico e intuitivo para cualquier usuario, sobre todo para los noveles en CEP, como es el uso de nodos y enlaces gráficos. Para lograrlo, se ha hecho uso del marco de trabajo

GMF, y su correspondiente marco de trabajo EMF para el modelado de los patrones, permitiéndose así que los modelos puedan ser compartidos entre diferentes herramientas de modelado, entre otras ventajas, tal y como se discute en el Capítulo 8. Por otra parte, dichos autores no mencionan ninguna solución software que hayan desarrollado para transformar los patrones de eventos tanto al código que se desplegará en un motor CEP como al código que se encargará de ejecutar las acciones correspondientes en un ESB.

Existen otros enfoques dirigidos por modelos para CEP que proporcionan lenguajes gráficos. No obstante, estos enfoques también quedan fuera del alcance de este estado del arte, debido a que se han creado con la finalidad de definir patrones de eventos restringidos a un dominio en particular. Por ejemplo, Decker *et al.* [Dec07] han propuesto un lenguaje gráfico, denominado BEMN (*Business Event Modeling Notation*), para modelar patrones de eventos en el contexto de los procesos del negocio.

2.2.2. Editores Gráficos para la Definición y la Generación de Código de Patrones de Eventos

Con el propósito de acercar la tecnología CEP a cualquier usuario no tecnólogo, este deberá disponer de algún editor gráfico que le permita definir de una forma gráfica e intuitiva las situaciones de interés que desea detectar en tiempo real en sus sistemas de información. Seguidamente, estos editores deberán permitir generar automáticamente el código que los implementa, siendo deseable que además este código se añada automáticamente tanto al motor CEP como al sistema o arquitectura encargado de ejecutar las acciones pertinentes; todo ello, de forma totalmente transparente para el usuario final.

Por este motivo, durante los últimos años se han propuesto algunos editores gráficos que tienen como objetivo principal lograr que sean utilizados por usuarios del negocio que no sean expertos ni en la tecnología CEP ni en ningún EPL en particular. No obstante, la gran mayoría de ellos requiere a este tipo de usuarios que escriba a mano, al menos, una parte del código EPL que implementa al patrón de eventos en cuestión.

Algunos de estos editores han sido construidos por empresas de desarrollo software y otros se han desarrollado dentro de grupos y/o proyectos de investigación [Blo14, Cha09, Cug12, Pro].

En cuanto a los editores creados por grandes empresas como herramientas complementarias a sus sistemas CEP particulares como, por ejemplo, Oracle CEP Visualizer [Ora14], StreamBase Studio [TIB13] y SAP Sybase ESP Studio [Syb14], estos suelen realizar únicamente transformaciones de los patrones al código específico que sea capaz de entender el sistema en cuestión. Esta dependencia entre motor CEP y editor podría requerir que el usuario deba definir n veces el mismo patrón de eventos para cada uno de los n distintos tipos de sistemas CEP en los que se necesite detectar el patrón. Lo que agrava todavía más esta situación es el hecho de que el usuario —o más bien el programador informático, debido a que normalmente se le exigirá escribir parte del código a mano— tendrá que invertir mucho tiempo en aprender cada uno de los lenguajes EPL, así como familiarizarse con el uso de cada uno de los editores correspondientes. Precisamente, el editor de patrones pre-

sentado en esta tesis persigue solventar todos estos problemas mencionados, minimizando la curva de aprendizaje.

A continuación, se describen los editores de patrones de eventos desarrollados dentro de grupos y/o proyectos de investigación.

Kalevar *et al.* [Kav10] han creado una interfaz web con dos funcionalidades principales: 1) la definición de los patrones de eventos y las acciones, así como la administración y la configuración del sistema CEP utilizado —concretamente, el sistema SARI (*Sense And Respond Infrastructure*) [Roz09, Sch05]—; y 2) la utilización de los patrones de eventos y las acciones, definidos previamente, con la intención de describir las situaciones de interés para un dominio en particular. Esta propuesta tiene varias limitaciones con respecto a la presentada en esta tesis doctoral. En primer lugar, esta herramienta exige que expertos en la tecnología CEP sean los que lleven a cabo tanto la definición de los patrones de eventos y las acciones, como la administración y configuración del sistema CEP. En segundo lugar, dicha definición se efectúa utilizando el lenguaje natural, lo que puede conllevar a definiciones de patrones y acciones ambiguas. En tercer lugar, existe una gran dependencia de los expertos en el negocio con respecto a los expertos en CEP, ya que los expertos en el negocio no podrán especificar cuáles son los patrones a detectar para un dominio en particular, ni tampoco las acciones, si estos no han sido creados previamente por los expertos en CEP. Además, las funcionalidades proporcionadas por este editor gráfico, así como su grado de usabilidad, son inferiores en gran medida de los proporcionados por el editor de patrones construido en esta tesis doctoral (véase más detalles sobre este editor de patrones en el Capítulo 5 y su evaluación en el Capítulo 8).

Sen *et al.* [Sen09] han desarrollado también un editor web, usando el marco de trabajo GWT (*Google Web Toolkit*) [Goo14], que hace posible la creación de patrones de eventos a partir de nodos de eventos, operadores y nodos de acción, así como la generación de código EPL. En este trabajo, se utiliza Twitter [Twi14] como productor de eventos, cuyos eventos producidos se transforman en formato RDF (*Resource Description Framework*) [W3C14b]. Seguidamente, estos eventos se convierten nuevamente para que puedan ser procesados por el motor Esper y, además, se obtienen automáticamente los tipos de estos eventos en formato RDFS. Finalmente, estos tipos de eventos se notifican al editor de patrones para que estén disponibles durante la creación de patrones de eventos. Aunque este editor ofrece ventajas con respecto al editor de Kalevar *et al.*, ya que en esta ocasión no se requiere la intervención de expertos en CEP como paso previo a la definición de patrones de eventos, no dispone de todas las funcionalidades proporcionadas por el editor de esta tesis y, además, la interfaz presenta limitaciones en cuento a usabilidad, puesto que los eventos, operadores y acciones se representan gráficamente como ventanas compuestas de botones y listas desplegables y es a través de estas ventanas donde el usuario final puede ir añadiendo las propiedades de los eventos que desea utilizar, así como las condiciones. A diferencia de esta propuesta, en el editor desarrollado en esta tesis el usuario puede seleccionar un tipo de evento de la paleta de herramientas y arrastrarlo hasta el área de trabajo, donde automáticamente se visualizará el evento con todas sus propiedades para las que podrá establecer condiciones directamente, enlazándolas con los operadores correspondientes.

Una nueva versión del editor creado por Sen *et al.*, en la que se incluyen nuevas funcionalidades, ha pasado a denominarse PANTEON [Sen12], formando parte del prototipo para la creación de patrones de eventos llevado a cabo dentro del proyecto europeo ALERT, ya mencionado previamente.

Por otro lado, CEP Editor [Soc13a], desarrollado dentro del proyecto SocEDA (*SOCial Event Driven Architecture*) [Soc13b] financiado por la agencia nacional francesa para la investigación o ANR (*L'Agence Nationale de la Recherche*), también se ha construido con el fin de definir patrones de eventos, así como transformarlos a código EPL de Esper.

Es importante destacar que de todos los mencionados, los editores PANTEON y CEP Editor son los únicos comparables con el propuesto en esta tesis doctoral por la similitud que presentan algunas de sus funcionalidades, no así en cuanto al grado de usabilidad que proporcionan, que es inferior. Una evaluación detallada sobre las ventajas e inconvenientes de cada uno de estos editores puede encontrarse en la Sección 8.2.2.

2.2.3. Propuestas para la Integración de CEP con EDA y/o SOA

Como se ha comentado en la Sección 2.1.4, una SOA 2.0 o ED-SOA consiste en una arquitectura software que combina SOA y EDA. Michelson [Mic06] fue una de las pioneras en proponer este tipo de arquitecturas, cuyo nexo de unión suele ser un ESB por su capacidad de simplificar considerablemente dicha integración [Mar06]. Por ejemplo, Juric [Jur10] ha propuesto una solución ED-SOA en la que los servicios actúan como productores y consumidores de eventos. Pero también existen algunas propuestas más antiguas en las que todavía no se hacía uso de un ESB, como es el caso del trabajo presentado por Yuan y Lu [Yua09].

Con la finalidad de detectar, en tiempo real, situaciones de interés en los sistemas de información, se han propuesto a lo largo de los últimos años algunos trabajos en los que se integra la tecnología CEP con EDA, otros trabajos que combinan CEP con SOA y, finalmente, otros que integran CEP con SOA 2.0.

Uno de los trabajos existentes que integra CEP con EDA es el propuesto por Dunkel *et al.* [Dun11], que facilita el proceso de la toma de decisiones ante situaciones relevantes que surjan en los sistemas de gestión del tráfico terrestre. Aunque usan el motor Esper para procesar la información en forma de eventos, una de las principales diferencias con la solución tecnológica propuesta en el Capítulo 6 es que no se hace uso de un ESB y, además, la arquitectura presenta cierta dependencia con el dominio donde se ha aplicado CEP; por ejemplo, tan solo se considera el uso de sensores como productores de eventos.

Entre las propuestas que integran CEP con SOA, destacan las siguientes. Sottara *et al.* [Sot09] proponen una arquitectura para una planta de tratamiento de aguas con la intención de detectar situaciones de interés durante la reducción o eliminación de la contaminación de las aguas que se reciben en dicha planta. En esta arquitectura se utiliza el ESB JBoss [JBo13a] para lograr dicha integración, y el motor de reglas Drools [JBo13b]. Zmuda *et al.* [Zmu10] proponen un marco de trabajo para la monitorización dinámica de entornos de ejecución SOA que utiliza tanto el ESB ServiceMix [Apa14c] como el motor Esper. Por otro lado, Romero et al. [Rom11] diseñan y evalúan una arquitectura para la

detección de situaciones críticas en el ámbito de la domótica que utiliza el modelo SCA (*Service Component Architecture*) [CSA14] —una tecnología que facilita la creación de servicios reutilizables dentro de una SOA— y el motor Esper.

Como puede comprobarse, todas estas arquitecturas que combinan CEP con SOA presentan diferencias con respecto a la propuesta en la tesis. En primer lugar, ninguna de ellas utiliza el ESB Mule, uno de los mejores ESB en la actualidad según Forrester [Rie14]. En segundo lugar, la propuesta de Sottara *et al.* no hace uso del motor Esper, uno de los motores CEP más utilizado en la actualidad por las ventajas que ofrece, como se ha mencionado en la Sección 2.1.5. En tercer lugar, algunas de las arquitecturas mencionadas son dependientes de un dominio en particular. Finalmente, aunque integran CEP con SOA, estas propuestas no se benefician de las ventajas que aportaría su integración con una EDA, descritas en la Sección 2.1.4. Lógicamente, esta integración podrá estar supeditada a las necesidades particulares de cada escenario de aplicación.

Una vez analizadas las propuestas que integran CEP con EDA o SOA, se describen los trabajos que integran CEP con SOA 2.0.

Bo *et al.* [Bo08] diseñan e implementan un ESB que permite la integración de una SOA 2.0 con un motor CEP. Este motor se ha implementado haciendo uso del álgebra relacional.

Asimismo, He *et al.* [He08] proponen su propia solución para la integración de CEP con SOA 2.0, implementada principalmente mediante Java, XML y tecnologías de servicios web. Esta solución es efectiva para la detección de situaciones anómalas ante las emisiones de los gases producidos por los tubos de escape de los vehículos. Como particularidad, esta arquitectura dispone de sensores de gases para medir dichas emisiones en tiempo real; en el caso de que las mediciones no sean las deseables, se usarán lectores RFID para identificar cuáles son los vehículos que suponen un problema para el medioambiente.

Zang *et al.* [Zan08] han propuesto también el uso de RFID en su propia SOA 2.0, además han implementado un prototipo de un motor CEP escrito en Java, denominado RTE-CEP.

Levina y Stantchev [Lev09] han propuesto una arquitectura ED-SOA con generadores de eventos, sensores de eventos y servicios de eventos como productores de eventos. Esta arquitectura, aplicada al dominio de la logística, se ha implementado con el marco de trabajo .NET 3.0 y, además, se ha hecho uso de Microsoft Workflow Foundation [Mic14] para gestionar los patrones de eventos que se pretenden detectar, así como los servicios que se encargarán de ejecutar las acciones correspondientes.

Wieland *et al.* [Wie09] también han creado una ED-SOA cuya característica principal es que los eventos complejos producidos por el motor Esper se notifican a un flujo BPEL (*Business Process Execution Language*) —el estándar OASIS (*Organization for the Advancement of Structured Information Standards*) [OAS07] que fundamentalmente permite definir procesos de negocio compuestos por varios servicios web— con el propósito de trasladar los efectos de las situaciones detectadas a los flujos de trabajo de los procesos del negocio.

Ravier [Rav10] ha desarrollado una SOA 2.0 con el fin de monitorizar las imágenes tomadas por videocámaras instaladas, por ejemplo, en lugares públicos, así como detectar

automáticamente intrusos en zonas no autorizadas, u otras situaciones anómalas. La combinación de EDA y SOA se ha logrado a través del uso de un ESB que se ha implementado con Java Business Integration Binding [Chr08], al que se le ha conectado el motor Esper.

Vidackovic y Weisbecker [Vid11] proponen una SOA 2.0 que dispone de un motor CEP, así como un motor de ejecución de procesos con la finalidad de facilitar la composición dinámica de servicios. En esta arquitectura, el ESB que utilizan es Open ESB [Com14] y el motor CEP es Esper.

Otros trabajos se han desarrollado en el seno de proyectos europeos. Por ejemplo, en el proyecto COMPAS (*Compliance-driven Models, Languages, and Architectures for Services*) del FP7 se ha propuesto una SOA 2.0 integrada con el motor Esper y el motor de ejecución de procesos Apache ODE [Apa14b], entre otros, mediante el uso de Apache ActiveMQ [Apa14a]. Téngase en cuenta que esta arquitectura se ha creado con el fin de monitorizar y garantizar el cumplimiento de los procesos de negocio en tiempo de ejecución. Por otra parte, en el proyecto ALERT perteneciente al mismo programa europeo, se propone el uso del ESB Petals [Pet14] para la integración de EDA y SOA con el motor ETALIS [Ani14].

Tras esta revisión pormenorizada de las propuestas existentes en la actualidad para la integración de CEP con SOA y EDA, no se ha encontrado ninguna que proporcione los beneficios de la solución tecnológica propuesta en el Capítulo 6.

En primer lugar, ninguna de ellas ha integrado el motor Esper con el ESB Mule, cuya elección se ha justificado previamente. Conviene recordar algunas de las ventajas de esta combinación: la capacidad de este motor de código abierto para procesar miles de eventos por segundo y dar soporte a la gestión dinámica de eventos en formato *maps*, dando una mayor flexibilidad a la hora de definir tipos de eventos con propiedades anidadas; y la capacidad de dicho ESB para integrarse con plataformas *cloud*, así como con múltiples herramientas y escenarios.

En segundo lugar, la mayoría de las propuestas no dispone de editores gráficos que ayuden, al experto en el dominio, a definir y generar automáticamente el código de los patrones de eventos que se requieran detectar así como las acciones que deban ejecutarse en un dominio concreto; excepto el editor construido dentro del proyecto ALERT, cuyas limitaciones se analizan detalladamente en el Capítulo 8.

Finalmente, aunque algunas de dichas arquitecturas presentan ciertas similitudes con la propuesta en esta tesis, atendiendo a algunas de sus funcionalidades, se han diseñado para dominios concretos, tales como monitorización del cumplimiento de los procesos de negocio, detección de intrusos mediante el análisis de grabaciones de video, detección de gases contaminantes emitidos al medioambiente; a diferencia de la propuesta en esta tesis que es independiente del dominio de aplicación.

Capítulo 3

Enfoque Dirigido por Modelos para el Procesamiento de Eventos Complejos en SOA 2.0

«La formulación de un problema es más importante que su solución.»
(Albert Einstein)

En este capítulo se propone un enfoque dirigido por modelos para el procesamiento de eventos complejos en SOA 2.0 que hace posible la definición de modelos de alto nivel, cercanos y comprensibles para los usuarios finales, siendo posteriormente transformados en código que pueda ser ejecutado en un motor CEP y en un ESB. Esto permitirá la detección de situaciones críticas o relevantes a partir de la información que fluya por sistemas complejos y heterogéneos, así como la notificación de las alarmas detectadas a los usuarios y sistemas interesados, todo esto de forma totalmente transparente al usuario final.

3.1. Motivación

Actualmente CEP es considerado por algunos expertos [Cha10, Etz10, Han13, Luc12] como una de las tecnologías claves que puede ser integrada con otras que necesitan procesar una ingente cantidad de información en tiempo real como *big data* o *fast data*.

Para poder procesar datos provenientes de fuentes heterogéneas, detectar situaciones relevantes a partir de estos y alertar sobre estas situaciones a los usuarios y sistemas interesados, se propone en esta tesis doctoral la integración de una SOA 2.0 con un motor CEP. Esta arquitectura estará compuesta básicamente por los productores de eventos, un ESB conectado con el motor CEP y los consumidores de eventos.

Esta solución, que será explicada en detalle en el Capítulo 6, permite que expertos tecnológicos puedan cumplir dichos objetivos; sin embargo, no es apta para que los usuarios

finales sin conocimientos de programación puedan satisfacer sus necesidades, ya que no tendrán el conocimiento ni la habilidad suficientes para definir en el sistema cuáles son las condiciones que deben cumplirse para detectar dichas situaciones en esta arquitectura, así como tampoco qué acciones tomar cuando estas sucedan.

Esto es así porque CEP requiere de expertos en el EPL proporcionado por el motor en cuestión. Como suele ocurrir en la mayoría de los casos, existen grandes dificultades para que los usuarios finales, que no son expertos tecnológicos, puedan beneficiarse del uso de la tecnología en cuestión.

Precisamente uno de los objetivos fundamentales de esta tesis doctoral es eliminar estas barreras existentes para que CEP esté al alcance de cualquier usuario que necesite detectar estas situaciones dentro de sus sistemas de información. Con este propósito, se ha propuesto un enfoque dirigido por modelos para el procesamiento de eventos complejos en arquitecturas orientadas a servicios y dirigidas por eventos.

3.2. Enfoque Propuesto

En esta sección se describe el enfoque dirigido por modelos propuesto cuyo propósito principal es la definición de modelos de alto nivel, cercanos y comprensibles para los usuarios finales, que serán a posteriori transformados en código que pueda ser ejecutado en un motor CEP y en un ESB. La Figura 3.1 muestra la representación gráfica de este enfoque.

Existen dos tipos de roles de usuario:

- *Experto en el dominio*: representa a las personas que tienen un vasto conocimiento con respecto a un área específica. Gracias a este conocimiento, estos expertos en una temática concreta son los candidatos ideales para definir con exactitud ese dominio donde CEP puede ser aplicado. Dicho de otro modo, estos expertos pueden definir cuáles son los tipos de eventos y sus propiedades que describen toda la información de un dominio en particular, asegurando que los modelos de dominio diseñados sean un fiel reflejo de la realidad.

- *Usuario final*: engloba a todas aquellas personas que tienen el conocimiento necesario para especificar las condiciones que deben cumplirse para detectar situaciones críticas y/o relevantes en un dominio concreto, pero no están familiarizados con ningún EPL para implementar estos patrones de eventos. Por tanto, se trata de los usuarios finales —o simplemente usuarios— que manejarán el editor propuesto en esta tesis doctoral para definir gráficamente estos patrones de eventos, abstrayéndoles de los detalles de implementación. Obviamente, un usuario final podrá ser, a su vez, un experto en el dominio.

Como puede observarse, este enfoque está compuesto de dos partes bien diferenciadas. La primera parte hace referencia a todo el proceso que se llevará a cabo en tiempo de diseño mientras que la segunda parte engloba el proceso que se desarrollará en tiempo de ejecución. A continuación, se enumera, siguiendo la misma numeración de la figura, la

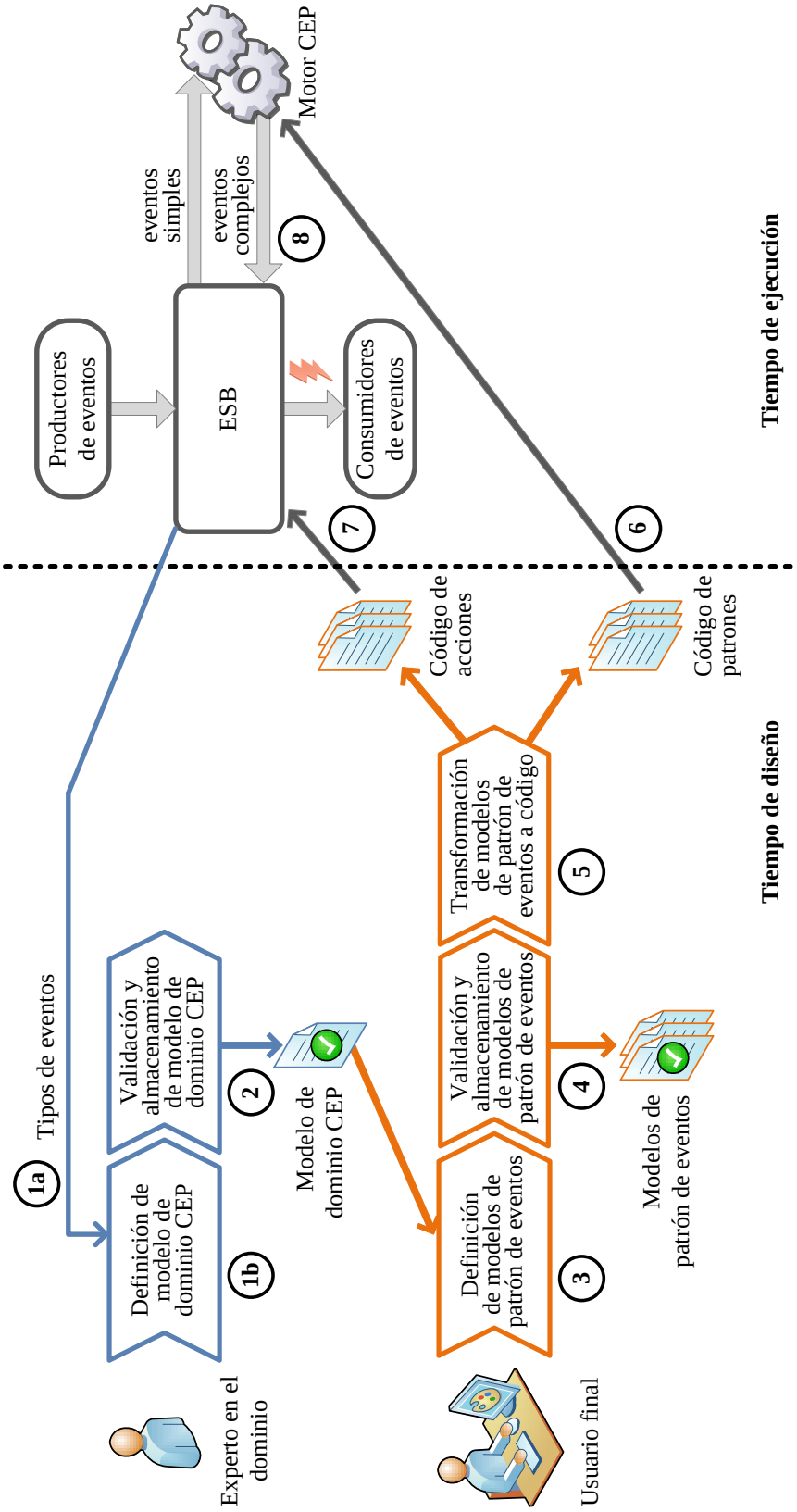

Figura 3.1: Enfoque dirigido por modelos para CEP en SOA 2.0.

secuencia de pasos implicados desde la definición del dominio donde se aplicará CEP hasta que el usuario final percibe una situación crítica o relevante:

1. *1a)* A partir de la información que llega al ESB proveniente de los productores de eventos, y siempre y cuando el experto en el dominio así lo solicite, el sistema generará automáticamente un modelo de dominio gráfico con los tipos de eventos y propiedades que fluyen por la arquitectura. El experto en el dominio podrá introducir modificaciones sobre este modelo de dominio generado automáticamente como, por ejemplo, añadir una descripción textual; *1b)* o bien el experto creará un modelo de dominio manualmente sin haberlo inferido previamente. Para ello, se propone tanto un lenguaje de modelado para la definición de dominios CEP como un editor gráfico de modelado de dominios CEP (véase el Capítulo 4).

2. Una vez definido el modelo de dominio CEP, el editor se encargará de validarlo. En caso de que no sea válido se avisará al experto en el dominio para que corrija los errores detectados. Este modelo será entonces guardado en el sistema y podrá exportarse e importarse para que pueda ser compartido y reutilizado por otros expertos en el dominio y/o usuarios finales (véase el Capítulo 4).

3. El usuario final, a partir del modelo de dominio CEP definido previamente, creará los modelos de patrones de eventos. Para ello, se propone tanto un lenguaje de modelado para la definición de patrones de eventos como un editor gráfico de modelado de patrones de eventos (véase el Capítulo 5). Téngase en cuenta que se trata de un editor gráfico de modelado para patrones de eventos que puede ser reconfigurado con cualquiera de los modelos de dominio CEP existentes.

4. Una vez definidos los modelos de patrón de eventos, el editor se encargará de validarlos. En caso de que alguno de ellos no sea válido se avisará al usuario para que corrija los errores detectados. Estos modelos serán entonces guardados en el sistema y podrán exportarse e importarse para que puedan ser compartidos y reutilizados por otros usuarios finales (véase el Capítulo 5).

5. Cada uno de estos modelos de patrón de eventos será transformado a código. En concreto, se generará automáticamente tanto el código del patrón correspondiente a las condiciones que deben cumplirse para detectar las situaciones críticas o relevantes dentro del motor CEP, como el código de las acciones a llevar a cabo en el ESB una vez se hayan detectado dichas situaciones (véase el Capítulo 5).

6. El código del patrón de eventos generado automáticamente será añadido al motor CEP en tiempo de ejecución (véase el Capítulo 6).

7. El código de las acciones generado automáticamente será añadido al ESB en tiempo de ejecución (véase el Capítulo 6).

8. A partir de los eventos simples que lleguen al motor CEP desde el ESB, y los patrones de eventos añadidos en tiempo de ejecución, el motor CEP creará nuevos eventos

complejos (alertas o alarmas) conforme se vayan detectando dichos patrones. Estos eventos complejos serán enviados al ESB que se encargará de difundirlos a todos los consumidores de eventos interesados en ellos —según se especifique en las acciones añadidas al ESB por cada patrón de eventos—, avisando al usuario final de las situaciones recién acontecidas (véase el Capítulo 6).

3.3. Conclusiones

En este capítulo se ha descrito un enfoque dirigido por modelos para el procesamiento de eventos complejos en SOA 2.0, en el que se proponen los dos tipos de usuarios que podrán interactuar con el sistema. Por un lado, los expertos en el dominio serán los encargados de modelar el dominio del que tienen un vasto conocimiento, así como de cerciorarse que los tipos de eventos y propiedades que componen este dominio sean los adecuados.

Por otro lado, los usuarios finales serán aquellos que, a partir de un modelo de dominio CEP definido, podrán diseñar los patrones de eventos que pretendan detectar en una SOA 2.0. Para ello, simplemente harán uso del editor de patrones desarrollado y el editor transformará los patrones al código que será desplegado tanto en el motor CEP como en el ESB; todo esto de forma transparente al usuario.

Capítulo 4

DSML y Editor Gráfico para la Definición de Dominios CEP

«No basta con adquirir sabiduría, es preciso además saber usarla.»
(Cicerón)

En este capítulo se propone un DSML con el propósito de unificar la descripción de los dominios CEP mediante el uso de modelos y de abstraer a los expertos en el dominio de los detalles de implementación necesarios para definir los tipos de eventos y propiedades que describen cada uno de estos dominios. Asimismo, se construye un editor gráfico para dicho DSML que permite a cualquier usuario definir fácilmente dominios CEP y validarlos automáticamente, así como exportarlos e importarlos para que puedan ser compartidos y reutilizados por otros expertos en el dominio.

4.1. Motivación

Como ya se ha mencionado previamente, CEP es una tecnología que permite procesar, analizar y correlacionar una ingente cantidad de información heterogénea en forma de eventos para detectar, en tiempo real, situaciones críticas o relevantes para un dominio concreto.

Algunos de los dominios más relevantes donde CEP ha sido aplicado son los siguientes: detección de fraude y seguridad informática [DC12, Gad12a, Gad12b, Gad13], domótica [BP14a, Rom11], salud [BP14b, Yua09], gestión del tráfico marítimo [BP12], energía y fabricación [Dun11], servicios basados en localización [Vik13], sistemas y operaciones financieros [Uhm11] e inteligencia de negocios operacional [Cha11]. Debido a la diversidad de dominios que actualmente pueden beneficiarse de la tecnología CEP, se considera necesario proporcionar un DSML que permita a los expertos en el dominio definir de una forma gráfica y amigable los dominios de interés para los que se desean detectar situaciones críticas en tiempo real. Es importante destacar que, además, dependiendo del motor CEP

que se utilice en cada caso, los tipos de eventos para un dominio de aplicación deberían implementarse atendiendo a la sintaxis particular del EPL ofrecido por el motor en cuestión. Esto conlleva dos grandes inconvenientes. Por un lado, un mismo dominio de interés tendría que implementarse tantas veces como el número de EPL proporcionados por los motores que se utilicen; traduciéndose en un incremento del tiempo requerido para definir el dominio CEP, así como un mantenimiento más costoso, debido a que una modificación del dominio CEP —por ejemplo, la adición de un nuevo tipo de eventos— requerirá la modificación de cada una de dichas implementaciones asociadas al mismo dominio. Por otra parte, cualquier definición de dominio que exija al usuario escribir manualmente parte del código que lo implementa, conllevaría inmediatamente a situar esta tarea fuera del alcance de cualquier usuario no informático.

Por todos estos motivos, en este capítulo se proporcionan los medios para facilitar la definición de los dominios CEP mediante modelos, gracias al uso de MDD. Esto facilitará la unificación de la descripción de los dominios, compuestos de los tipos de eventos que fluyen por el sistema de información objeto de análisis, así como de las características o propiedades de cada uno de estos. Es más, con vistas a poner al alcance de cualquier usuario experto en el dominio, no necesariamente experto en tecnologías, la definición de dichos dominios, se propone y construye también en este capítulo un editor gráfico para el modelado de dominios CEP. Este software proporciona un entorno amigable e intuitivo con el que este tipo de usuarios podrá diseñar gráficamente estos dominios de forma totalmente transparente y sin necesidad de tener ningún conocimiento previo sobre ningún EPL en particular, así como compartirlos con otros usuarios.

4.2. Lenguaje de Modelado Específico de Dominio

En esta sección, se propone un DSML para la definición de dominios CEP, detallándose tanto la sintaxis abstracta como la sintaxis concreta de este lenguaje.

4.2.1. Sintaxis Abstracta

La sintaxis abstracta de un DSML se compone de un metamodelo, donde se especifican los conceptos del lenguaje y las relaciones entre estos, así como las restricciones sobre los elementos del modelo y sus relaciones para velar por el cumplimento de las reglas del dominio. Así pues, en esta sección se describe el metamodelo de dominios CEP propuesto en esta tesis doctoral junto con las reglas que permiten validar que los modelos de dominio CEP están bien formados.

Metamodelo de Dominio CEP

Seguidamente, se describe el metamodelo propuesto para definir un dominio CEP. La Figura 4.1 muestra las metaclases de este metamodelo y sus relaciones que se describen a continuación:

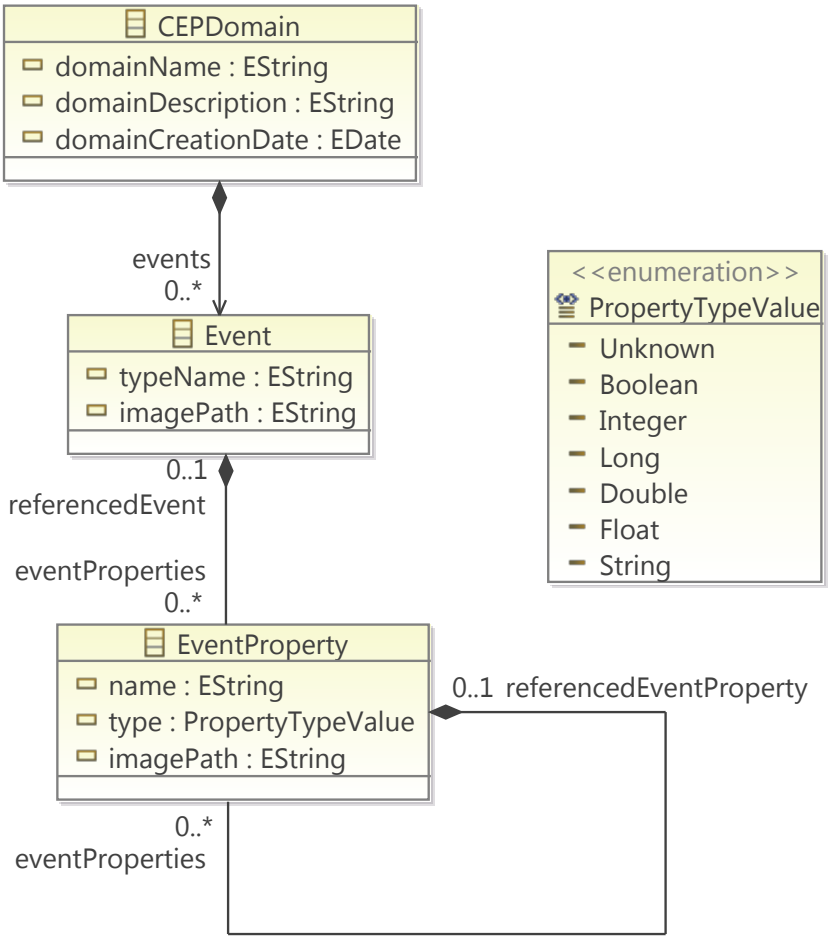

Figura 4.1: Metamodelo de dominio CEP.

- *CEPDomain*: se trata de la metaclase principal del metamodelo, por tanto, la raíz de cada modelo será una instancia de *CEPDomain* que representa un dominio CEP concreto compuesto por uno o más tipos de eventos (*Event*). Para cada dominio deberá especificarse su nombre (*domainName*), una descripción textual (*domainDescription*) y la fecha de creación (*domainCreationDate*).

- *Event*: describe un evento para un dominio CEP en particular. Cada evento pertenecerá a un tipo concreto (*typeName*) y, adicionalmente, tendrá asociado una imagen que lo represente gráficamente, cuya localización (*imagePath*) deberá ser especificada. Además, cada evento estará formado por una o más propiedades de eventos (*EventProperty*). Por ejemplo, en el ámbito de la domótica, *EventoHogar* podría ser un tipo de evento que determine que la información proviene de sensores localizados en distintos hogares, mientras que sensor, localización, marca de tiempo, temperatura interior y humedad interior, podrían ser algunas de las propiedades de dicho tipo de evento. Estas propiedades describirán, en concreto, cuál es el sensor del que proviene la información y dónde se encuentra, en qué momento se ha tomado la información, así como la temperatura y humedad interiores de un hogar, respectivamente.

- *EventProperty*: representa la propiedad o característica de un evento. Cada propiedad debe tener un nombre (*name*) y uno de los siguientes tipos (*type*): *Unknown*, *Boolean, Integer*, *Long*, *Double*, *Float* o *String*. Al igual que en el caso del evento, la propiedad podrá tener asociada una imagen que la represente gráficamente, cuya localización (*imagePath*) deberá ser especificada. Además, una propiedad puede, a su vez, contener una o más propiedades, esto es, se permite definir propiedades de eventos anidadas. Como ejemplo podría definirse la propiedad localización compuesta de tres propiedades: nombre, latitud y longitud.

Restricciones del Metamodelo de Dominio CEP

A continuación, se especifican cuáles son las restricciones definidas para este metamodelo con el propósito de extender y completar la semántica del mismo. Así pues, en la Tabla 4.1 se presentan las reglas de validación que debe cumplir cualquier modelo conforme al metamodelo, describiéndose las restricciones impuestas para cada metaclase.

Tabla 4.1: Restricciones del metamodelo de dominio CEP.

Metaclase	Restricción
CEPDomain	El nombre del dominio (*domainName*) debe ser especificado.
	Debe contener, al menos, un evento (*Event*).
Event	El nombre del tipo de evento (*typeName*) debe ser especificado.
	Debe contener, al menos, una propiedad de evento (*EventProperty*).
EventProperty	El nombre de la propiedad (*name*) debe ser especificado.

(Continúa en la página siguiente)

(Continúa de la página anterior)

Metaclase	Restricción
	En el caso de que se trate de una propiedad anidada, esta no podrá contener varias propiedades con el mismo nombre (*name*) si se encuentran en el mismo nivel de anidamiento.
	En el caso de que se trate de una propiedad anidada, esta no deberá tener asociado ningún tipo de propiedad (*type*), puesto que el tipo será determinado por las propiedades que contenga.

4.2.2. Sintaxis Concreta

La sintaxis concreta de un DSML permite establecer una relación entre los conceptos del metamodelo y su representación textual o gráfica.

Por ello, además de la definición del metamodelo de dominio CEP y sus restricciones, también se ha creado una notación gráfica propia para cada uno de los elementos que el usuario deberá utilizar cuando desee diseñar un modelo de dominio CEP. Esta sintaxis concreta de los modelos de dominio CEP se presenta en la Tabla 4.2.

Tabla 4.2: Sintaxis concreta del metamodelo de dominio CEP.

Nombre	Notación
Event	
EventProperty	

4.3. Editor Gráfico

En esta sección se describe el método de desarrollo que se ha seguido para construir el editor gráfico de dominios CEP. Este método se compone de los siguientes 5 pasos:

1. **Creación del metamodelo de dominios CEP**: consiste en la implementación del metamodelo que define el DSML de dominios CEP propuesto en la Sección 4.2.1. Para ello, el metamodelo se expresa textualmente con Emfatic y se construye con el lenguaje de metamodelo Ecore.

2. **Generación del editor gráfico por defecto**: a partir del metamodelo implementado en el paso anterior y utilizando GMF y EuGENia, se genera automáticamente una versión inicial del editor gráfico.

3. **Personalización del editor generado**: dada las limitaciones, en cuanto a funcionalidad y usabilidad, que presenta el editor GMF generado automáticamente en el paso 2, en este paso se personaliza el editor modificando su paleta de herramientas, así como añadiendo un menú de opciones para que el usuario final pueda llevar a cabo el modelado de dominios CEP de forma intuitiva.

4. **Implementación de las reglas de validación**: se implementan las reglas que permitirán validar si un dominio CEP modelado está bien formado. Para ello, se emplea el lenguaje EVL del proyecto Epsilon.

5. **Creación de una aplicación Eclipse RCP**: se crea una aplicación Eclipse RCP (*Rich Client Platform*) [Vog13] que integra todos los *plugins* obtenidos en los pasos anteriores, de forma que esta aplicación —el editor de dominios CEP— pueda ser ejecutada fuera del IDE (*Integrated Development Environment*) Eclipse y en múltiples plataformas.

A continuación, se describe pormenorizadamente cada uno de estos pasos.

4.3.1. Creación del Metamodelo de Dominios CEP

El primer paso consiste en la implementación del metamodelo de dominios CEP (véase la Sección 4.2.1). Para cumplir este propósito, se ha utilizado el lenguaje Emfatic, una notación textual para describir metamodelos basados en EMF, añadiendo a la especificación del metamodelo anotaciones GMF. Estas anotaciones facilitarán posteriormente la generación del editor gráfico, ya que determinan qué elementos del metamodelo se podrán modelar con el editor y cuál será su representación gráfica.

El Listado 4.1 muestra la implementación del metamodelo utilizando Emfatic. A continuación, se explican las anotaciones GMF más relevantes:

- `gmf.diagram`: indica qué clase es la raíz del metamodelo. Además, se han especificado las siguientes opciones:
  - `model.extension`: la extensión del modelo de dominio (modelo EMF), denominada como *domain*.
  - `diagram.extension`: la extensión del fichero de diagrama (diagrama), que se ha denominado como *domain_diagram*.
- `gmf.node`: especifica que la instancia de la clase correspondiente será representada como nodo en el diagrama. Además, se han especificado las siguientes opciones:
  - `figure`: el tipo de figura que representará el nodo. En este caso se trata de un rectángulo.
  - `label`: el texto que será utilizado como etiqueta del nodo.
  - `border.color`: el color RGB (*Red Green Blue*) del borde del nodo.
  - `resizable`: si se permite o no modificar el tamaño del nodo.
  - `tool.name`: el nombre de la herramienta de la paleta del editor que debe ser creada.
  - `tool.description`: la descripción de la herramienta de la paleta del editor que debe ser creada.

- `gmf.compartment`: indica la creación de un contenedor donde podrán alojarse los elementos del modelo que sean conformes al tipo de la referencia asociada. En este caso el contenedor se tratará como una lista de elementos, puesto que así ha sido definido con la opción `layout="list"`.

Listado 4.1: Definición del metamodelo de dominios CEP utilizando la notación textual Emfatic para metamodelos basados en EMF.

```
@namespace(uri="www.uca.es/modeling/cep/domain", prefix="domain")
package domain;

@gmf.diagram(model.extension="domain", diagram.extension="domain_diagram")
class CEPDomain {
    attr String domainName;
    attr String domainDescription;
    attr Date domainCreationDate;
    val Event[*] events;
}

@gmf.node(figure="rectangle", label="typeName", border.color="110,110,110",
    tool.name="Event", tool.description="Add an event", resizable="false")
class Event {
    attr String typeName;
    attr String imagePath;
    @gmf.compartment(layout="list")
    val EventProperty[*]#referencedEvent eventProperties;
}

@gmf.node(figure="rectangle", label="name", border.color="110,110,110", tool
    .name="Event Property", tool.description="Add an event property",
    resizable="false")
class EventProperty {
    attr String name;
    attr PropertyTypeValue type;
    attr String imagePath;
    ref Event#eventProperties referencedEvent;
    @gmf.compartment(layout="list")
    val EventProperty[*]#referencedEventProperty eventProperties;
    ref EventProperty#eventProperties referencedEventProperty;
}

enum PropertyTypeValue {
    Unknown;
    Boolean;
    Integer;
    Long;
    Double;
    Float;
    String;
}
```

4.3.2. Generación del Editor Gráfico por Defecto

A partir del metamodelo descrito con Emfatic, y enriquecido con anotaciones específicas de GMF, se ha utilizado EuGENia para generar automáticamente los tres modelos requeridos por GMF para implementar un editor gráfico: el modelo que describe los elementos gráficos (*gmfgraph*), el modelo que define la paleta de herramientas del editor (*gmftool*) y el modelo que establece las correspondencias entre los modelos anteriores (*gmfmap*).

Gracias al uso de EuGENia, se han llevado a cabo automáticamente las siguientes acciones:

1. La generación del metamodelo en formato Ecore a partir del definido con Emfatic.

2. La generación del modelo *genmodel* desde el modelo Ecore. El modelo *genmodel* contendrá el código generado asociado al metamodelo Ecore.

3. La generación del código EMF a partir del modelo *genmodel*. Esto generará tres proyectos: *es.uca.modeling.cep.domain.edit* —contendrá el código requerido para poder manipular los modelos de dominio CEP—, *es.uca.modeling.cep.domain.editor* —contendrá un *plugin* que proporciona un editor de modelos EMF basado en árboles— y *es.uca.modeling.cep.domain.tests* —con el código necesario para realizar pruebas unitarias. Además, el proyecto principal de partida, *es.uca.modeling.cep.domain*, se ha convertido en un proyecto Java con el código correspondiente a cada una de las clases del metamodelo.

4. La generación de los modelos *gmfgraph*, *gmftool* y *gmfmap*.

5. La generación del modelo *gmfgen*, esto es, el modelo de generación de código para GMF.

6. La generación del código correspondiente al editor GMF de dominios CEP, denominado *es.uca.modeling.cep.domain.diagram*, a partir del modelo *gmfgen*.

Tras la finalización de estas acciones, se ha obtenido un editor GMF generado automáticamente por EuGENia que puede ejecutarse dentro del IDE Eclipse.

4.3.3. Personalización del Editor Generado

Aunque la herramienta EuGENia facilita la implementación del editor GMF para manejar modelos de dominio CEP, el editor generado en el paso anterior presenta limitaciones desde el punto de vista de la funcionalidad y la usabilidad, sobre todo, teniendo en cuenta el tipo de usuario que lo usará. Por este motivo, se ha personalizado tanto la paleta de herramientas como el menú de opciones del editor, tal y como se describirá en las siguientes subsecciones.

En la Figura 4.2 puede observarse el aspecto del editor gráfico de dominios CEP, tras su personalización, en el que puede distinguirse sus partes principales:

- Una paleta de herramientas (panel derecho) desde la que el experto en el dominio seleccionará los elementos que desea incorporar a sus modelos.
- Un área de trabajo (panel central) donde podrá modelar los tipos de eventos y sus propiedades para un dominio CEP.
- Un menú (barra de menús superior) que le permitirá seleccionar fácilmente la acción que desea ejecutar entre las disponibles sobre dominios CEP.
- Una vista de propiedades (panel inferior) para añadir o modificar la información asociada a los distintos elementos del modelo como, por ejemplo, el tipo (*type*) de la propiedad nombre (*name*) que se encuentra contenida dentro de la propiedad localización (*location*).

Paleta de Herramientas

La paleta del editor está compuesta por dos herramientas, tal y como se ha definido en el metamodelo con las anotaciones GMF:

- *Event*: la herramienta que permitirá insertar un tipo de evento en el área de trabajo del editor.
- *EventProperty*: la herramienta que permitirá definir propiedades para los eventos que hayan sido insertados previamente en el área de trabajo del editor. En el caso de que sea necesario definir una propiedad anidada, deberá insertarse dentro de otra propiedad.

En la Sección 4.2.2 se detalló la sintaxis concreta del DSML para la definición de dominios CEP, lo que ha requerido la personalización de la paleta del editor. Concretamente, tanto a la herramienta *Event* como a la herramienta *EventProperty* se les han asociado por defecto un icono en verde con la letra E y P, respectivamente. Sin embargo, estos iconos podrán ser sustituidos por otras imágenes a petición del experto en el dominio, indicando la ruta de la imagen en el atributo *imagePath*. Esta funcionalidad extra implementada dota al editor de mayor usabilidad en tanto en cuanto los tipos de eventos y sus propiedades pueden ser identificados mediante imágenes con las que el propio usuario esté familiarizado.

Conviene recordar que el área de trabajo es la parte del editor donde los elementos correspondientes a las herramientas de la paleta pueden ser insertados para definir los modelos que conformen al metamodelo de dominios CEP. Los atributos de estos elementos pueden ser modificados tanto gráficamente como mediante la vista de propiedades del editor. En este editor solamente puede utilizarse la herramienta *Event* directamente sobre el área de trabajo, ya que las propiedades de eventos siempre deberán ser añadidas dentro de alguno de estos eventos. De hecho, si el usuario final intenta utilizar directamente *EventProperty* sobre el área de trabajo, el editor se lo impedirá.

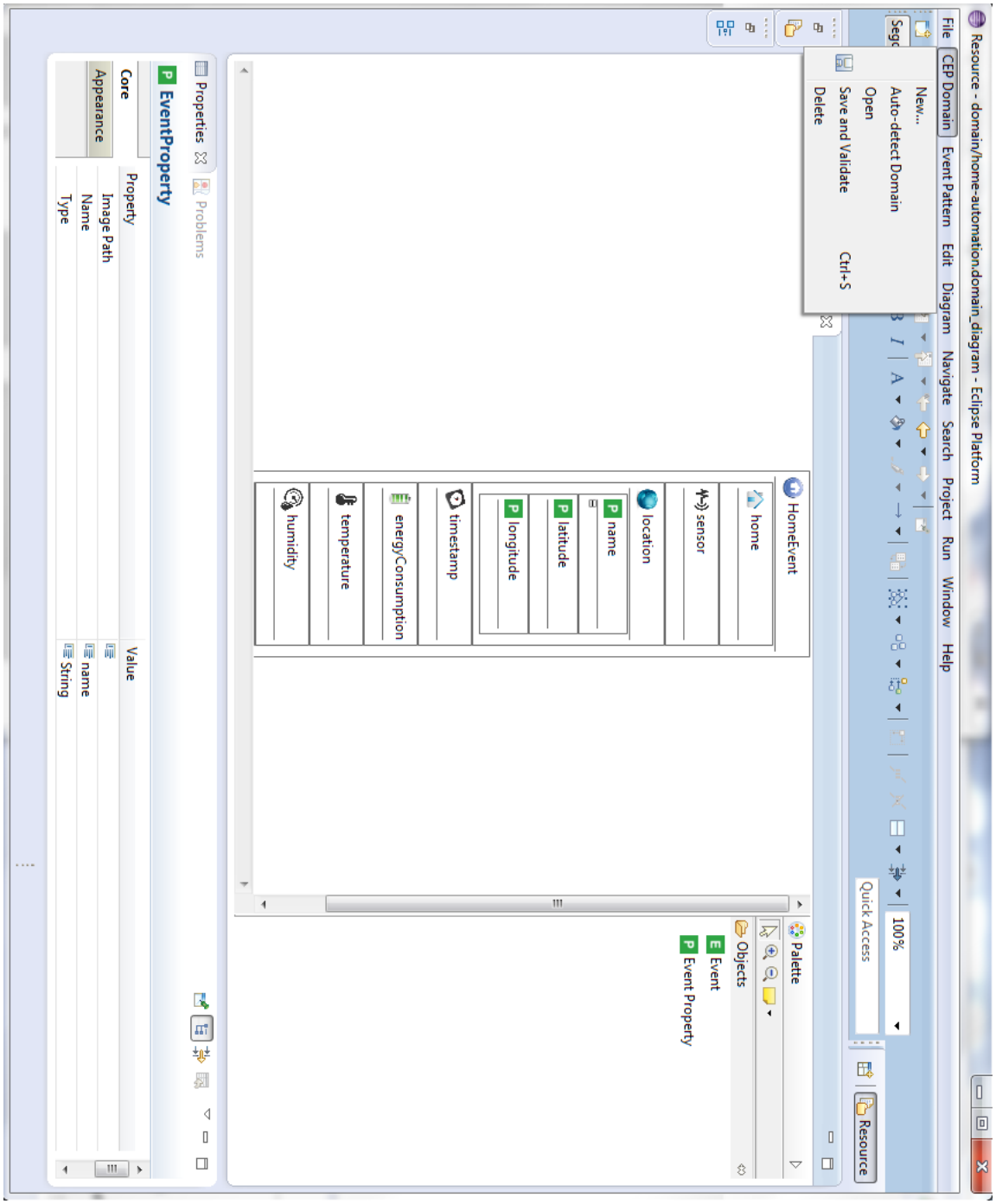

Figura 4.2: Editor gráfico construido para el modelado de dominios CEP.

Menú de Opciones

El editor de dominios CEP dispone de la barra de menús disponible en toda aplicación Eclipse. Además de los menús por defecto, se ha añadido el menú *CEP Domain* para dotar de mayor facilidad de uso al editor. En particular las opciones que se han añadido son las siguientes:

- *New*: esta opción permite crear un nuevo dominio CEP, facilitando al usuario la creación del modelo que represente dicho dominio.
- *Auto-detect Domain*: esta opción permitirá obtener automáticamente el dominio CEP —los tipos de eventos y sus propiedades— a partir de los eventos que fluyan por la arquitectura SOA 2.0 que esté conectada con este editor. Esta funcionalidad será explicada con más detalle en el Capítulo 6.
- *Open*: esta opción permite abrir uno de los dominios CEP que hayan sido previamente definidos.
- *Save and Validate*: esta opción permitirá al experto en el dominio tanto guardar el modelo de dominio que se encuentre activo como validarlo. En caso de que el modelo no esté bien formado, se indicarán cuáles son los problemas encontrados, además de indicarse gráficamente con un icono de error en los elementos implicados del modelo.
- *Delete*: esta opción permitirá eliminar uno de los dominios CEP previamente descritos.

Por otro lado, también se han añadido dos opciones más al menú *File* de la barra de menús:

- *Import CEP Domain*: esta opción permitirá importar otros dominios CEP que hayan sido definidos previamente con este editor. Para ello, el usuario deberá indicar el archivo a importar cuya nomenclatura seguirá el siguiente formato: *nombre_domain.zip.*
- *Export CEP Domain*: esta opción se utilizará para exportar los dominios que hayan sido modelados por el usuario, siempre y cuando los modelos hayan sido validados. Si el usuario ha utilizado imágenes concretas para los tipos de eventos y las propiedades, entonces dichas imágenes también serán exportadas a tamaño 20 x 20 píxeles. Así pues, se obtendrá un archivo cuya nomenclatura seguirá el siguiente formato: *nombre_domain.zip*. Este archivo contendrá tanto el modelo de dominio (modelo EMF) y el fichero de diagrama (diagrama) como las imágenes a las que haga referencia dicho modelo.

4.3.4. Implementación de las Reglas de Validación

Las restricciones del metamodelo definidas en la Sección 4.2.1 se han implementado usando EVL. Aunque OCL es el estándar *de facto* para implementar las restricciones en los lenguajes de modelado, tiene varias limitaciones con respecto a EVL. Por un lado, OCL

no proporciona la personalización de mensajes que puedan ser reportados al usuario en el caso de que se produzca algún fallo. Por otro lado, OCL trata todos los fallos como errores sin hacer ninguna distinción entre errores (*errors*) –críticos— y avisos (*warnings*) —no críticos—, además un invariante no puede depender de otros invariantes puesto que son considerados como unidades independientes. Otro de los inconvenientes del uso de OCL con respecto a EVL es que no proporciona al usuario la posibilidad de reparar de forma semi-automática inconsistencias que presenten los modelos. Un ejemplo de este tipo de reparaciones podría ser solicitar al usuario el nombre de un evento al que todavía no se le ha indicado su tipo, encargándose el editor de asociar dicho nombre al atributo *typeName* del evento correspondiente.

En el Listado 4.2 se muestra un extracto de las restricciones implementadas con EVL para validar los modelos de dominio CEP. Estas especificaciones de validación se organizan en un módulo EVL que contiene un conjunto de invariantes agrupados por el contexto donde pueden ser aplicados. En este caso, el contexto *Event* especifica que el tipo de instancias *Event* será para el que se evalúen los invariantes *HasEventName* y *HasProperty*. Cada invariante EVL está formado por un nombre y el cuerpo (*check*) donde se define la comprobación a realizar que puede ser de dos tipos: *constraint* —captura errores críticos que invalidan el modelo— y *critique* —captura situaciones no críticas que, aunque no invaliden el modelo, deberían ser corregidas para mejorar la calidad del modelo. Opcionalmente puede definirse una guarda (*guard*) que limita su aplicación a un subconjunto de instancias del tipo definido por dicho contexto. En este ejemplo, *guard* comprueba si se define el nombre del tipo de evento. A continuación, se describe el mensaje (*message*) que se desea devolver al usuario describiendo el porqué ha fallado la restricción. Finalmente, en un invariante puede definirse opcionalmente reparaciones (*fixes*) que facilitarán al usuario la reparación semi-automática de las inconsistencias identificadas.

Listado 4.2: Extracto de las restricciones implementadas con EVL para validar los modelos de dominio CEP.

```
[...]
context Event {

    constraint HasEventName {

        check : self.typeName.isDefined() and self.typeName.trim().length()
           > 0

        message : 'Event must have type name.'

        fix {

            title : "Set the value for 'typeName'."

            do {
                var name : String;
                name = UserInput.prompt("Please enter the value for '
                   typeName':");
```

```
                self.typeName = name;
            }
        }
    }

    constraint HasProperty {

        guard : self.satisfies("HasEventName")

        check : self.eventProperties.size() > 0

        message : "Event '" + self.typeName + "' must contain at least one
            EventProperty."

    }

}
[...]
```

4.3.5. Creación de una Aplicación Eclipse RCP

Tras la finalización de los pasos anteriores, se han obtenido unos *plugins*, los componentes de software de los que está compuesta la aplicación Eclipse del editor gráfico de dominios CEP desarrollada. Sin embargo, esta aplicación requiere ser ejecutada desde dentro del propio IDE Eclipse, lo que impediría que el editor estuviese al alcance de cualquier usuario no informático.

Con el propósito de ofrecer un editor gráfico que pueda ser ejecutado fuera de Eclipse, así como en múltiples plataformas —Windows, Linux y Mac, entre otras—, se ha creado una aplicación Eclipse RCP. Para llevar a cabo esta tarea, se ha creado un producto Eclipse que integra dichos *plugins* obtenidos, más los recursos (imágenes, iconos, etc.) que formen la aplicación. Además, este producto se ha exportado haciendo uso de DeltaPack, un paquete Eclipse que contiene las características específicas de plataforma y los *plugins* requeridos para construir y exportar una aplicación Eclipse para múltiples plataformas.

4.4. Modelado de Dominios CEP con el Editor

En el Capítulo 3 se ha propuesto un enfoque dirigido por modelos para el procesamiento de eventos complejos en SOA 2.0. Gracias al editor gráfico de modelado de dominios CEP presentado en este capítulo, el experto en el domino ya podrá abordar la ejecución de las fases implicadas durante dicho modelado (véase la Figura 4.3):

- *Definición del modelo de dominio CEP*: el experto en el dominio será el responsable de definir gráficamente el dominio CEP para un escenario concreto, como puede ser la salud, la bolsa, la domótica, la seguridad informática, entre otros. El modelo obtenido será conforme al metamodelo descrito en la Sección 4.2.1. Como ya se comentó al

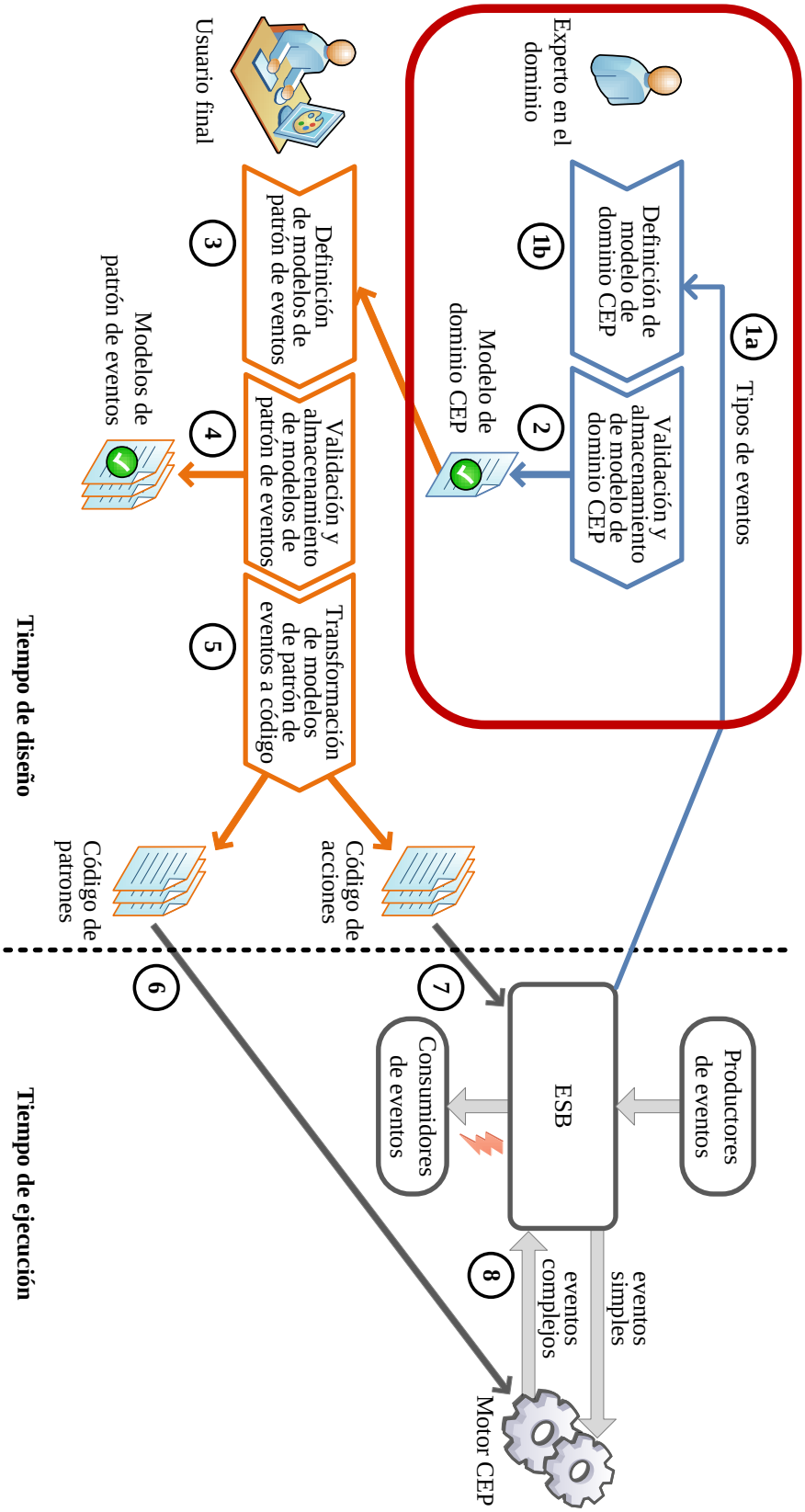

Figura 4.3: Enfoque dirigido por modelos para CEP en SOA 2.0: modelado de dominios CEP.

describir este enfoque, los tipos de eventos y propiedades que describen dicho dominio podrán ser inferidos automáticamente desde los eventos que fluyan por la SOA 2.0. Para llevar a cabo esta primera fase, el experto en el dominio deberá usar, al menos, una de las siguientes opciones del menú del editor: crear un nuevo dominio CEP desde cero (*CEP Domain > New*) o autodetectar el dominio (*CEP Domain > Auto-detect Domain*).

- *Validación y almacenamiento del modelo de dominio CEP*: una vez que el dominio haya sido modelado, este será validado comprobándose que se cumplen las restricciones definidas en la Sección 4.2.1; además, será guardado el modelo, junto con su diagrama, en el sistema y podrá exportarse e importarse para que el dominio CEP pueda ser compartido y reutilizado por otros expertos en el dominio y/o usuarios finales. En esta fase, deberá usarse la opción de guardar y validar el modelo (*CEP Domain > Save and Validate*) y, en el caso de que se desee importar o exportar, también deberían usarse las opciones *File > Import CEP Domain* o *File > Export CEP Domain*, respectivamente.

Conviene destacar que en el Capítulo 7 se presentan dos casos de estudio en los que se realizan estas fases con la finalidad de modelar dos dominios CEP sobre domótica y seguridad en redes.

4.5. Conclusiones

En este capítulo se ha proporcionado un DSML para unificar la descripción de dominios CEP, representándolos como modelos, facilitando así al experto en el dominio la definición de estos sin necesidad de conocer detalles específicos sobre la tecnología CEP. En concreto, se ha propuesto tanto un metamodelo de dominio CEP junto con las restricciones que permiten validar los modelos conformes al metamodelo, como su notación gráfica. Este metamodelo permite describir cuáles son los tipos de eventos y sus propiedades para cualquier dominio donde pueda aplicarse CEP.

Asimismo, se ha creado un editor GMF con el fin de facilitar a los usuarios un entorno amigable e intuitivo con el que podrán definir gráficamente dominios CEP, sin necesidad de tener ningún conocimiento previo sobre la tecnología CEP. Además, este editor validará automáticamente estos modelos de dominio comprobando que sean conformes a su metamodelo.

Uno de los principales beneficios que brinda este editor es la eliminación de las barreras existentes en la actualidad, donde los expertos en el dominio requieren de la ayuda de expertos en CEP para poder definir dichos dominios sobre los que desean detectar situaciones críticas. Es importante destacar que este editor posibilita no solo que el experto en el dominio pueda diseñar y validar los modelos de dominio CEP, sino que también pueda exportarlos para su uso por otros expertos y/o empresas así como importar los dominios facilitados por terceros.

Capítulo 5

DSML y Editor Gráfico Reconfigurable para la Definición y la Generación de Código de Patrones de Eventos

«El objetivo de la abstracción es no perderse en vaguedades y crear un nuevo nivel semántico en el que se pueda ser absolutamente preciso.»
(Edsger Dijkstra)

En este capítulo se propone un DSML con el propósito de unificar la descripción de los patrones de eventos mediante el uso de modelos y de abstraer a los usuarios finales de los lenguajes requeridos para implementarlos. Asimismo, se construye un editor gráfico para dicho DSML que permite a cualquier usuario definir fácilmente estos patrones, validarlos y generar automáticamente el código que los implementa, así como exportarlos e importarlos para que puedan ser compartidos y reutilizados por otros usuarios.

5.1. Motivación

Para poder detectar en tiempo real situaciones de interés utilizando la tecnología CEP, es necesario implementar los patrones de eventos haciendo uso del EPL que sea proporcionado por el motor CEP encargado de dicha detección. Por lo cual, se requeriría, al usuario que pretenda llevar a cabo esta tarea, un buen grado de experiencia en dicho lenguaje. Esto provoca que los usuarios que tienen un vasto conocimiento en el dominio para el que se necesitan definir los patrones de eventos, pero que son inexpertos en CEP, no tengan la posibilidad de usar todo este conocimiento con el fin de realizar una rigurosa definición de todas las condiciones que deban cumplirse para detectar dichas situaciones. En este contexto, un estudio efectuado por la empresa ebizQ [BEA07] concluye que un 84 % de

los encuestados consideran que la definición de estos patrones de eventos debe realizarse por expertos en el dominio, que son los que realmente disponen de todo el conocimiento necesario para ello, frente a un 16 % que estiman más conveniente que sea llevado a cabo por programadores. La encuesta realizada al amparo de esta tesis también corrobora esta necesidad (véase la Sección 8.2).

Por otra parte, tal y como se ha reflejado en el enfoque dirigido por modelos para CEP en SOA 2.0 descrito en el Capítulo 3, para detectar situaciones críticas o relevantes en tiempo real se necesita no solo el motor CEP que sea capaz de detectar las situaciones en tiempo real, sino también un ESB que, entre otras cosas, hará posible que la información proveniente de fuentes heterogéneas pueda enviarse al motor para su procesamiento, así como notificar las situaciones detectadas por el motor a los usuarios interesados. Esto implicaría la creación manual de código como ya se ha mencionado para el motor CEP y, además, para el ESB en cuestión; lo que indudablemente deja a estas tecnologías fuera del alcance de cualquier usuario no informático.

Teniendo en cuenta estos problemas y dada la cantidad de EPL que existen en la actualidad con los que los programadores pueden implementar estos patrones, en este capítulo se propone un DSML con el propósito de definir un *lenguaje común* para que cualquier usuario pueda describir fácilmente un patrón, independientemente del lenguaje que se requiera para ser ejecutado en el motor CEP en cuestión. Gracias a las técnicas de MDD, este patrón definido como un modelo podrá transformarse posteriormente a código de distintos EPL con las consecuentes ventajas: oculta los aspectos técnicos de los EPL a los usuarios finales, y la productividad así como la calidad del software mejoran puesto que los modelos son más fáciles de mantener y el código generado automáticamente estará libre de errores. Es más, con el propósito de acercar estas tecnologías a cualquier usuario, eliminando las barreras existentes en cuanto a la programación requerida, se propone y desarrolla también en este capítulo un editor gráfico intuitivo y fácil de usar para el modelado de patrones de eventos, así como su transformación automática tanto al código de los patrones que debe ser capaz de comprender el motor CEP, como al código de las acciones que deberán ser desplegadas en el ESB. De esta forma, la notificación temprana de las situaciones en las que cada usuario no tecnólogo esté interesado en detectar para un dominio en particular, se convertirá en una realidad.

5.2. Lenguaje de Modelado Específico de Dominio

En esta sección, se propone un DSML para la definición de patrones de eventos, detallándose tanto la sintaxis abstracta como la sintaxis concreta de este lenguaje.

5.2.1. Sintaxis Abstracta

Seguidamente, se describe el metamodelo de patrones de eventos propuesto en esta tesis doctoral junto con las reglas que permiten validar los modelos de patrón de eventos que son conformes a dicho metamodelo.

Metamodelo de Patrones de Eventos

A continuación, se describe el metamodelo para definir patrones de eventos de una forma gráfica e intuitiva. La Figura 5.1 muestra las principales metaclases de este metamodelo y sus relaciones que se describen recorriendo la figura de arriba abajo y de izquierda a derecha:

- *CEPEventPattern*: se trata de la metaclase principal del metamodelo, por tanto, la raíz de cada modelo será una instancia de *CEPEventPattern* que representa un patrón de eventos que puede contener enlaces (*Link*) —para establecer relaciones entre el resto de componentes—, elementos (*EventPatternElement*) —necesarios para definir las condiciones a detectar por el patrón—, un evento complejo (*ComplexEvent*) —el tipo de evento a crear cada vez que se detecte el patrón— y acciones (*Action*) —que deberán llevarse a cabo una vez detectado el patrón. Para cada patrón de eventos deberá especificarse su nombre (*patternName*), una descripción textual (*patternDescription*), el nombre del dominio al que pertenece este patrón (*domainName*), la fecha de creación (*patternCreationDate*), y si el patrón ha sido o no transformado a código y desplegado en una SOA 2.0 (*patternDeployed*).

- *Link*: define la representación gráfica de una o más relaciones entre operandos (*Operand*) y operadores (*Operator*). Para cada enlace establecido entre un operando y un operador debe especificarse cuál es el orden (*order*) de ese operando con respecto al resto de operandos conectados a ese operador. El valor de *order* debe estar comprendido entre *1* y *N*, siendo *N* el número total de operandos conectados al operador.

- *Operand*: describe uno de los datos necesarios para llevar a cabo la operación con la que esté enlazado. Existen operandos de condición (*ConditionOperand*) —aquellos que pueden ser enlazados con los operadores de condición— y operandos de patrón (*PatternOperand*) —aquellos que pueden ser enlazados con los operadores de patrón. La Figura 5.2 detalla las metaclases *ConditionOperand* y *PatternOperand*, y la Tabla 5.1 describe textualmente los tipos de operandos. Téngase en cuenta que para posibilitar la definición de patrones más complejos, un operador (*Operator*) podrá ser, a su vez, un tipo de operando, que permitirá enlazarse con otro operador. Esta es la razón por la que en dicha figura el operador de agregación (*AggregationOperator*) es un tipo de operando. Asimismo, un evento complejo (*ComplexEvent*) también es considerado como un tipo de operando, ya que podría ser necesario enlazarlo con alguna de las acciones (*Action*) que deban llevarse a cabo tras su detección.

- *Operator*: representa una operación especifica que puede llevarse a cabo con uno o más operandos —dependiendo de la aridad del operador. Existen operadores de condición (*ConditionOperator*) —aquellos que pueden ser enlazados con los operandos de condición—, operadores de patrón (*PatternOperator*) —aquellos que pueden ser enlazados con los operandos de patrón— y operadores de agregación (*AggregationOperator*) —las funciones de agregación que pueden aplicarse sobre ciertos operandos.

La Figura 5.3 detalla las metaclases *ConditionOperator*, *PatternOperator* y *AggregationOperator*. Para simplificar esta figura no se muestra la aridad —unario, binario o n-ario— de cada operador; esta información se especifica en la Tabla 5.2 donde se describen textualmente los tipos de operadores.

- *UnaryOperator*: define un operador unario, que es aquel que debe ser enlazado con un único operando.
- *BinaryOperator*: define un operador binario, que es aquel que debe ser enlazado con dos operandos.
- *NaryOperator*: define un operador n-ario, que es aquel que debe ser enlazado con dos o más operandos.
- *EventPatternElement*: representa los elementos que pueden ser usados con el objetivo de definir el patrón de eventos propiamente dicho. Estos elementos pueden de dos tipos: *EventPatternCondition* —las condiciones que deben cumplirse para detectar un determinado patrón de eventos— y *DataWindow* —si la búsqueda de las condiciones se aplicará solamente a un subconjunto de eventos. *EventPatternCondition* es, a su vez, clasificado en cinco categorías: *ConditionOperand*, *PatternOperand*, *ConditionOperator*, *PatternOperator* y *AggregationOperator*. La Figura 5.4 muestra la metaclase *DataWindow* y la Tabla 5.3 describe textualmente los tipos de ventanas de datos.
- *ComplexEvent*: describe el tipo de evento complejo a crear cada vez que se detecte el patrón. El evento complejo pertenecerá a un tipo concreto (*typeName*) y, adicionalmente, tendrá asociado una imagen que lo represente gráficamente, cuya localización (*imagePath*) deberá ser especificada. El nombre del tipo (*typeName*) coincidirá con el nombre del patrón de eventos (*patternName* en *CEPEventPattern*); de esta forma, el nombre del evento complejo determinará cuál es el patrón que lo ha detectado. Además, cada evento complejo estará formado por una o más propiedades de evento complejo (*ComplexEventProperty*).
- *Action*: determina la acción que deberá llevarse a cabo cuando un evento complejo sea creado al detectarse el patrón de eventos correspondiente. Estas acciones se clasifican en dos categorías: *Email* —para enviar un evento complejo por correo electrónico— y *Twitter* —para enviarlo a una cuenta de Twitter. La Figura 5.5 muestra la metaclase *Action* y la Tabla 5.4 describe textualmente los tipos de acciones. Aunque actualmente solo se han definido estos dos tipos de acciones, este metamodelo puede ser extendido fácilmente para incluir nuevas acciones conforme se vayan necesitando; para ello, simplemente será necesario crear la metaclase asociada a la nueva acción y que extienda a la metaclase *Action*.

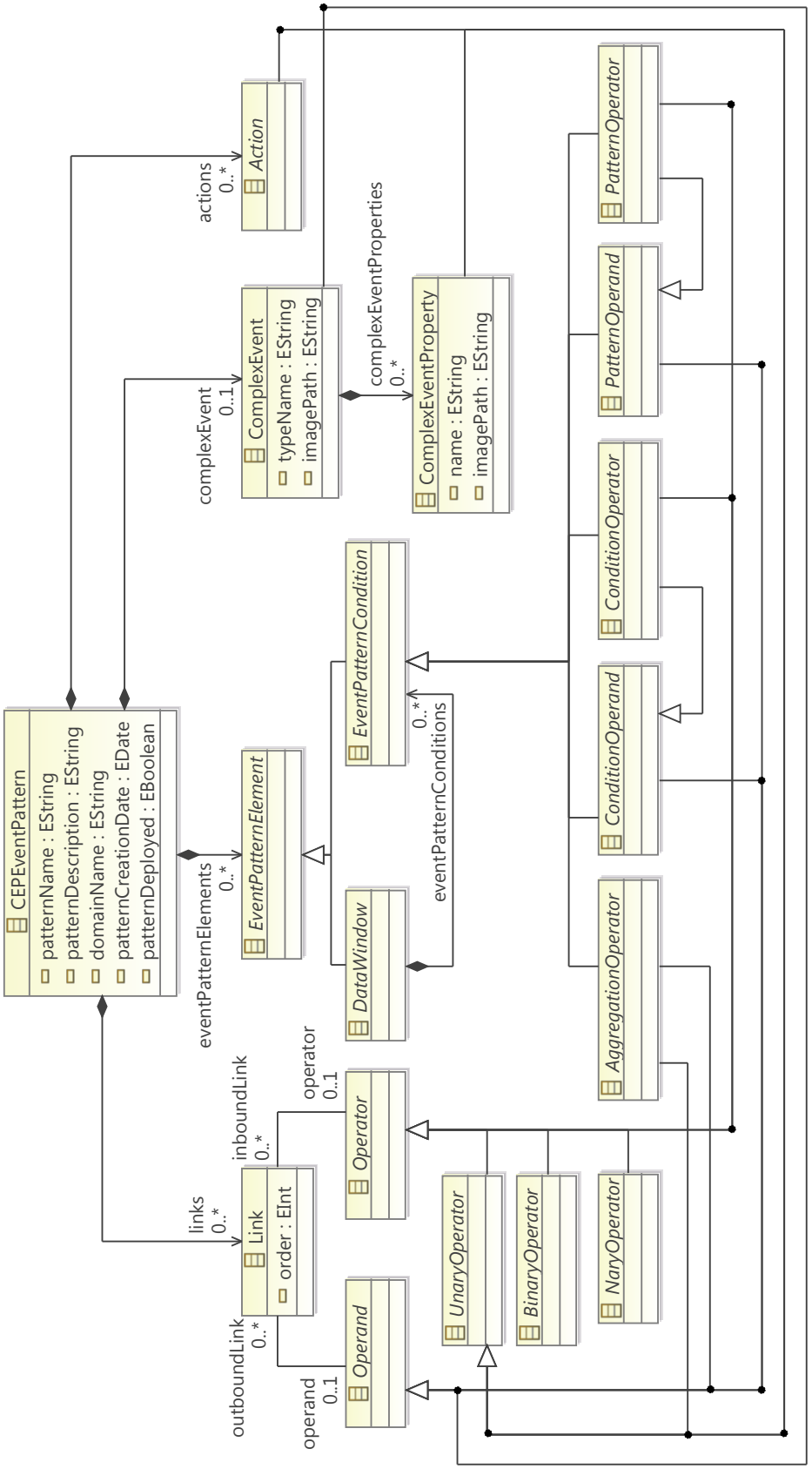

Figura 5.1: Metamodelo de patrones de eventos.

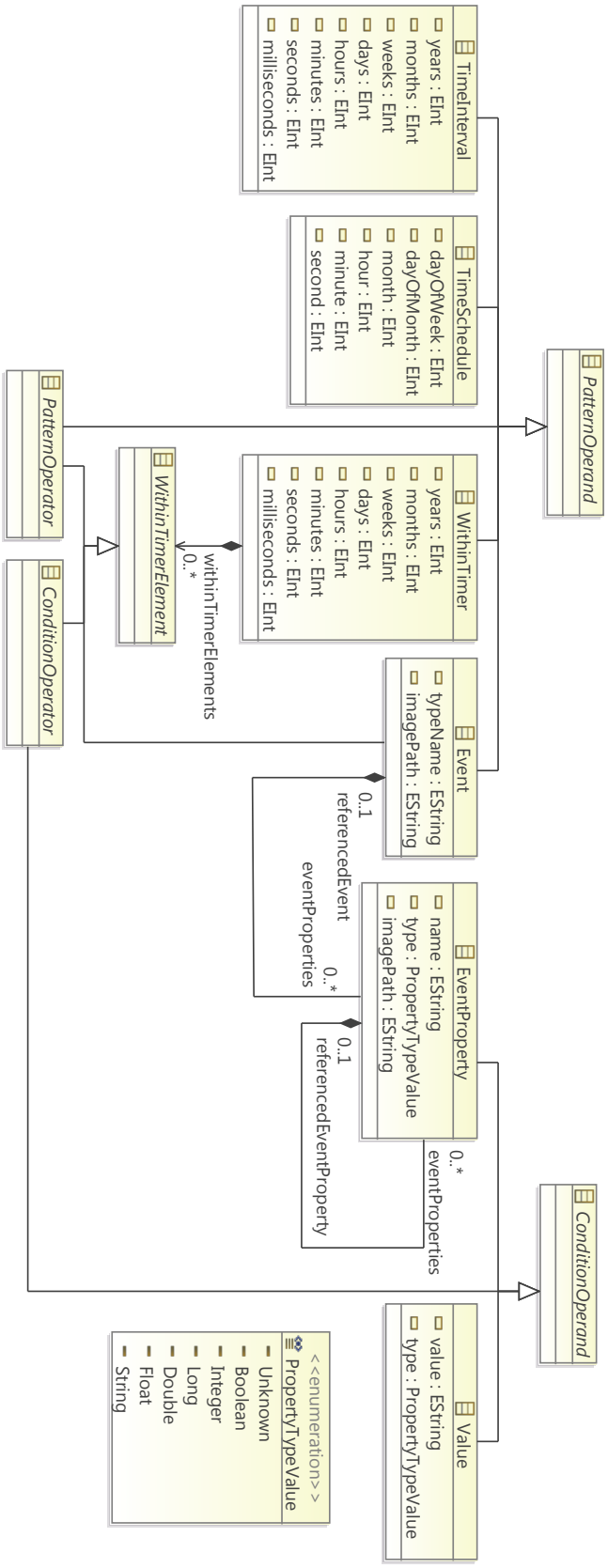

Figura 5.2: Metamodelo de patrones de eventos: operandos.

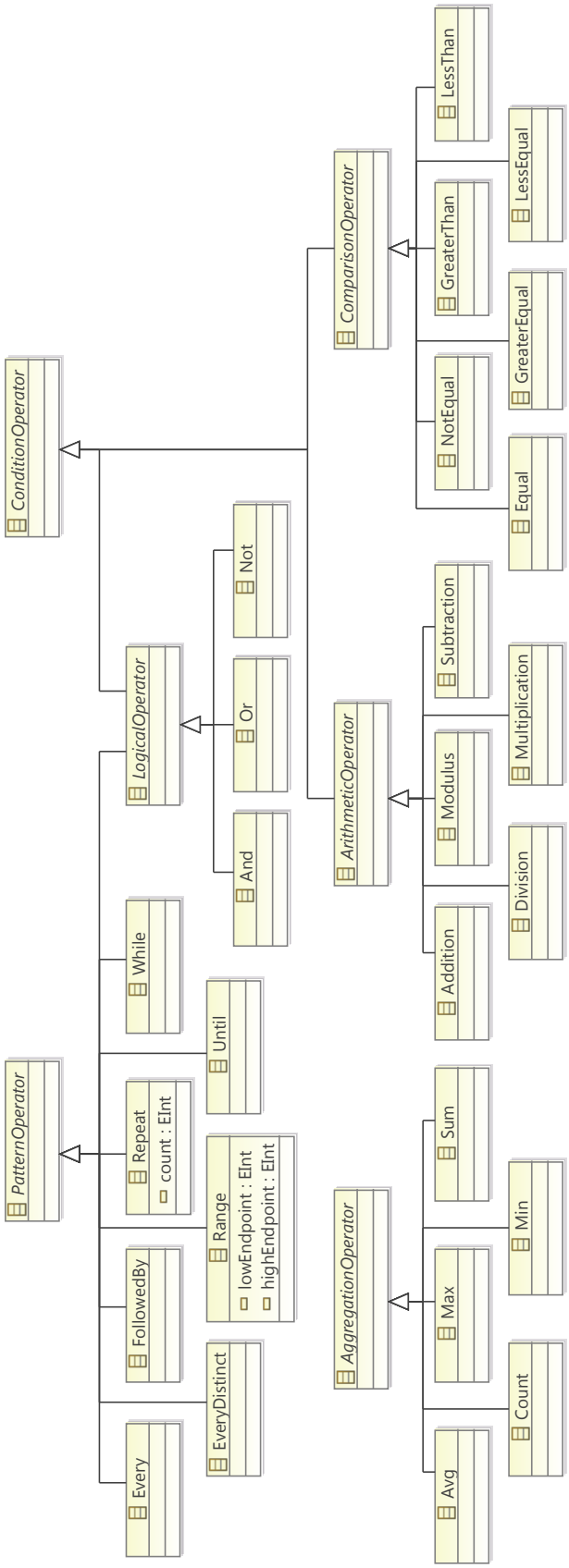

Figura 5.3: Metamodelo de patrones de eventos: operadores.

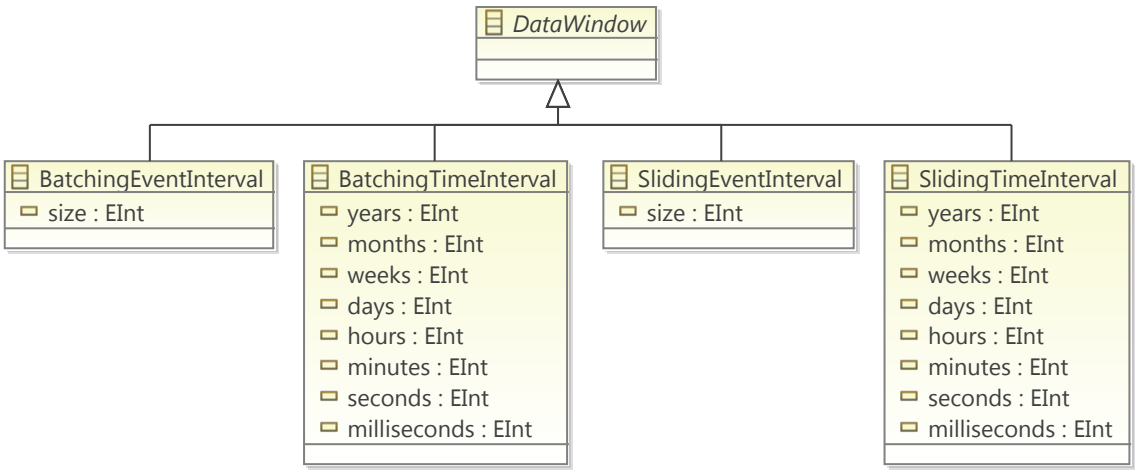

Figura 5.4: Metamodelo de patrones de eventos: ventanas de datos.

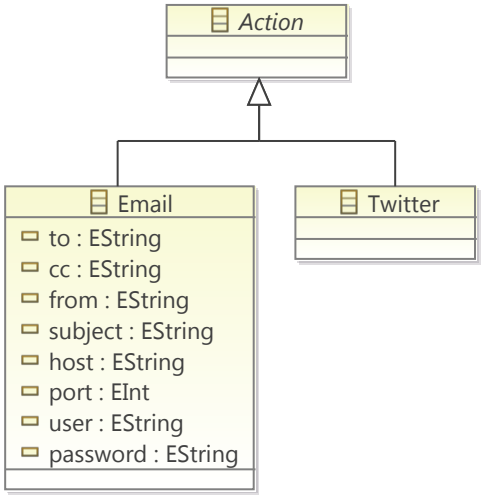

Figura 5.5: Metamodelo de patrones de eventos: acciones.

Tabla 5.1: Operandos del metamodelo de patrones de eventos.

Tipo	Operando	Descripción
Pattern	*TimeInterval*	Se espera el período de tiempo especificado (*years*, *months*, *weeks*, *days*, *hours*, *minutes*, *seconds*, *milliseconds*) antes de activarse.
	TimeSchedule	Se activa cuando se cumple el tiempo especificado (*dayOfWeek*, *dayOfMonth*, *month*, *hour*, *minute*, *second*).
	WithinTimer	Se activa si la expresión de patrón que contiene llega a ser cierta durante el período de tiempo especificado (*years*, *months*, *weeks*, *days*, *hours*, *minutes*, *seconds*, *milliseconds*).
Pattern & WithinTimerElement	*Event*	Describe un evento para un dominio CEP en particular. Cada evento pertenecerá a un tipo concreto (*typeName*) y, adicionalmente, tendrá asociado una imagen que lo represente gráficamente, cuya localización (*imagePath*) deberá ser especificada. Además, cada evento estará formado por una o más propiedades de eventos (*EventProperty*).
Condition	*EventProperty*	Representa la propiedad o característica de un evento. Cada propiedad debe tener un nombre (*name*) y uno de los siguientes tipos (*type*): *Unknown*, *Boolean*, *Integer*, *Long*, *Double*, *Float* o *String*. Al igual que en el caso del evento, la propiedad podrá tener asociada una imagen que la represente gráficamente, cuya localización (*imagePath*) deberá ser especificada. Además, una propiedad puede, a su vez, contener una o más propiedades, esto es, se permite definir propiedades de eventos anidadas.
	Value	Especifica un valor (*value*) de uno de los siguientes tipos (*type*): *Boolean*, *Integer*, *Long*, *Double*, *Float* o *String*.

Tabla 5.2: Operadores del metamodelo de patrones de eventos.

Tipo	Subtipo	Operador	Aridad	Descripción
Pattern		*Every*	U	Selecciona cada evento del tipo especificado.
		EveryDistinct	U	Similar a *Every* pero elimina los resultados duplicados según la expresión indicada en *distinct-value*.

(Continúa en la página siguiente)

(Continúa de la página anterior)

Tipo	Subtipo	Operador	Aridad	Descripción
		FollowedBy	N	Una expresión de patrón es la causa de otra expresión.
		Range	U	Especifica el número mínimo (*lowEndpoint*) y el número máximo (*highEndpoint*) de veces que debe ocurrir una expresión de patrón.
		Repeat	U	Especifica el número exacto (*count*) de veces que debe ocurrir una expresión de patrón.
		Until	B	Se detecta una expresión de patrón hasta que la condición (otra expresión de patrón) se evalúa como verdadera.
		While	B	Se detecta una expresión de patrón mientras la condición (otra expresión de patrón) se evalúa como verdadera.
Pattern & Condition	*Logical*	*And*	N	Devuelve un valor verdadero solo si todos los operandos son verdaderos.
		Or	N	Devuelve un valor verdadero si al menos uno de los operandos es verdadero.
		Not	U	Devuelve un valor verdadero si el operando es falso, y un valor falso si el operando es verdadero.
Condition	*Comparison*	*Equal*	B	Devuelve un valor verdadero si operando1 = operando2.
		GreaterEqual	B	Devuelve un valor verdadero si operando1 $\geq$ operando2.
		GreaterThan	B	Devuelve un valor verdadero si operando1 $>$ operando2.
		LessEqual	B	Devuelve un valor verdadero si operando1 $\leq$ operando2.
		LessThan	B	Devuelve un valor verdadero si operando1 $<$ operando2.
		NotEqual	B	Devuelve un valor verdadero si operando1 $\neq$ operando2.
	Arithmetic	*Addition*	B	Suma dos valores numéricos.
		Division	B	Divide un valor numérico por otro.

(Continúa en la página siguiente)

(Continúa de la página anterior)

Tipo	Subtipo	Operador	Aridad	Descripción
		Modulus	B	Devuelve el módulo de dos valores numéricos.
		Multiplication	B	Multiplica dos valores numéricos.
		Subtraction	B	Resta dos valores numéricos.
Aggregation		*Avg*	U	Devuelve la media de los valores de una expresión.
		Count	U	Devuelve el número de los valores de una expresión.
		Max	U	Devuelve el máximo de los valores de una expresión.
		Min	U	Devuelve el mínimo de los valores de una expresión.
		Sum	U	Devuelve la suma de los valores de una expresión.

Tabla 5.3: Ventanas de datos del metamodelo de patrones de eventos.

Ventana de datos	Descripción
BatchingTimeInterval	Cada ventana es un intervalo de un período de tiempo fijo (*years*, *months*, *weeks*, *days*, *hours*, *minutes*, *seconds*, *milliseconds*). Los eventos a lanzar se disparan cuando finaliza la ventana correspondiente.
BatchingEventInterval	Cada ventana es un intervalo de un número de eventos fijo (*size*). Los eventos a lanzar se disparan cuando finaliza la ventana correspondiente.
SlidingTimeInterval	Cada ventana es un intervalo de un período de tiempo fijo (*years*, *months*, *weeks*, *days*, *hours*, *minutes*, *seconds*, *milliseconds*) y se desplaza en intervalos regulares relativos a los demás.
SlidingEventInterval	Cada ventana es un intervalo de un número de eventos fijo (*size*) y se desplaza en intervalos regulares relativos a los demás.

Tabla 5.4: Acciones del metamodelo de patrones de eventos.

Acción	Descripción
Email	Especifica la dirección o direcciones de correo electrónico donde se desea enviar el evento complejo detectado. Deberá especificarse, al menos, el emisor del correo (*from*), el receptor o receptores del correo (*to*), así como el host (*host*), el puerto (*port*), el nombre de usuario (*user*) y su contraseña (*password*). Opcionalmente, podrá incluirse el asunto (*subject*) y los destinatarios en copia (*cc*).
Twitter	Especifica la cuenta de Twitter donde se registrarán todas las situaciones acontecidas (eventos complejos).

Restricciones del Metamodelo de Patrones de Eventos

A continuación, se especifican cuáles son las restricciones definidas para este metamodelo con el propósito de extender y completar la semántica del mismo. Así pues, en la Tabla 5.5 se presentan las reglas de validación que debe cumplir cualquier modelo conforme al metamodelo, describiéndose las restricciones impuestas para cada metaclase.

Tabla 5.5: Restricciones del metamodelo de patrones de eventos.

Metaclase	Restricción
CEPEventPattern	El nombre del patrón de eventos (*patternName*) debe ser especificado.
	Debe contener, al menos, un evento (*Event*) o una ventana de datos (*DataWindow*).
	Debe incluir el evento complejo (*ComplexEvent*).
	Si se usa algún operador de patrón (*PatternOperator*) entonces podrá utilizarse a lo sumo una ventana de datos (*DataWindow*) que contendrá todas las condiciones del patrón.
Link	No pueden enlazarse dos operadores iguales.
	No pueden enlazarse propiedades que estén anidadas.
ComplexEvent	El nombre del tipo de evento complejo (*typeName*) debe ser especificado.
	Debe contener, al menos, una propiedad de evento complejo (*ComplexEventProperty*).
	Las propiedades de evento complejo deben ser únicas.
ComplexEventProperty	El nombre de la propiedad (*name*) debe ser especificado.
	Debe ser enlazada por una propiedad de evento (*EventProperty*), o un operador aritmético (*ArithmeticOperator*) o de agregación (*AggregationOperator*).
Event	El nombre del tipo de evento (*typeName*) debe ser especificado.

(Continúa en la página siguiente)

(Continúa de la página anterior)

Metaclase	Restricción
	Debe contener, al menos, una propiedad de evento (*EventProperty*).
EventProperty	El nombre de la propiedad (*name*) debe ser especificado.
	Debe estar contenida en un evento (*Event*) o en una propiedad de evento (*EventProperty*).
	En el caso de que se trate de una propiedad anidada, esta no podrá contener varias propiedades con el mismo nombre (*name*) si se encuentran en el mismo nivel de anidamiento.
Operator	No puede tener enlaces de entrada con el mismo origen.
UnaryOperator	Debe tener un enlace de entrada.
BinaryOperator	Debe tener dos enlaces de entrada.
	El orden de los enlaces debe estar comprendido entre 1 y 2.
NaryOperator	Debe tener, al menos, dos enlaces de entrada.
	El orden de los enlaces debe estar comprendido entre 1 y N, siendo N el número total de operandos conectados al operador.
AggregationOperator	Debe enlazar una propiedad de evento complejo (*ComplexEventProperty*).
EveryDistinct	El primer enlace de entrada debe enlazar un evento (*Event*), y el segundo enlace una propiedad de evento (*EventProperty*).
While	El primer enlace de entrada debe enlazar un operador lógico (*LogicalOperator*) o un operador *Every*, *EveryDistinct* o *FollowedBy*, y el segundo enlace un operador lógico o de comparación (*ComparisonOperator*).
Range	El valor de *lowEndpoint* debe ser menor o igual al valor de *highEndpoint*.
	Debe enlazar un operador *Until* con orden 1.
Repeat	El valor de *count* debe ser positivo.
TimeInterval	Al menos uno de sus atributos (*years*, *months*, *weeks*, *days*, *hours*, *minutes*, *seconds*, *milliseconds*) debe ser positivo.
TimeSchedule	Al menos uno de sus atributos (*dayOfWeek*, *dayOfMonth*, *month*, *hour*, *minute*, *second*) debe ser positivo.
	El atributo *dayOfWeek* debe estar comprendido entre 0 (domingo) y 6 (sábado), *dayOfMonth* entre 1 y 31, *month* entre 1 y 12, *hour* entre 0 y 23, *minute* entre 0 y 59, y *second* entre 0 y 59.
WithinTimer	Al menos uno de sus atributos (*years*, *months*, *weeks*, *days*, *hours*, *minutes*, *seconds*, *milliseconds*) debe ser positivo.
DataWindow	Debe contener al menos un evento (*Event*).
BatchingEventInterval	El atributo *size* debe ser positivo.

(Continúa en la página siguiente)

(Continúa de la página anterior)

Metaclase	Restricción
BatchingTimeInterval	Al menos uno de sus atributos (*years*, *months*, *weeks*, *days*, *hours*, *minutes*, *seconds*, *milliseconds*) debe ser positivo.
SlidingEventInterval	El atributo *size* debe ser positivo.
SlidingTimeInterval	Al menos uno de sus atributos (*years*, *months*, *weeks*, *days*, *hours*, *minutes*, *seconds*, *milliseconds*) debe ser positivo.
ArithmeticOperator	Los dos operandos deben ser numéricos: *Integer*, *Long*, *Double* y *Float*.
ComparisonOperator	Los dos operandos deben ser del mismo tipo: *Boolean*, *Integer*, *Long*, *Double*, *Float* y *String*.
Value	Debe especificarse un valor.
	Si el valor es de tipo *Boolean*, entonces debe ser *true* o *false*.
	Debe enlazar un operador lógico (*LogicalOperator*), de comparación (*ComparisonOperator*) o aritmético (*ArithmeticOperator*).
Email	El atributo *to* debe tener al menos una dirección de correo electrónico con formato válido.
	El atributo *from* debe tener al menos una dirección de correo electrónico con formato válido.
	Si se especifica el atributo *cc*, deberá tener al menos una dirección de correo electrónico con formato válido.
	El atributo *port* debe estar comprendido entre 0 y 65535. Los puertos más comunes son 25, 465, 475, 587 y 2525.
	Debe especificarse un valor para *host*, *user* y *password*.

5.2.2. Sintaxis Concreta

Además de la definición del metamodelo de patrones de eventos, también se ha creado una notación gráfica propia para cada uno de los elementos que el usuario final deberá utilizar cuando desee diseñar un modelo de patrones de eventos. Esta sintaxis concreta de los modelos de patrón de eventos se presenta en la Tabla 5.6.

Tabla 5.6: Sintaxis concreta del metamodelo de patrones de eventos.

Categoría	Nombre	Notación
	Link	
	Value	
Simple Events	*Event*	
	EventProperty	
Complex Events	*ComplexEvent*	

(Continúa en la página siguiente)

(Continúa de la página anterior)

Categoría	Nombre	Notación
	ComplexEventProperty	
Pattern Timers	*TimeInterval*	
	TimeSchedule	
	WithinTimer	
Pattern Operators	*Every*	
	EveryDistinct	
	FollowedBy	
	Range	
	Repeat	
	Until	
	While	
Logical Operators	*And*	
	Not	
	Or	
Comparison Operators	*Equal*	
	GreaterEqual	
	GreaterThan	
	LessEqual	
	LessThan	
	NotEqual	
Arithmetic Operators	*Addition*	
	Division	
	Modulus	
	Multiplication	
	Subtraction	
Aggregation Operators	*Avg*	

(Continúa en la página siguiente)

(Continúa de la página anterior)

Categoría	Nombre	Notación
	Count	
	Max	
	Min	
	Sum	
Data Windows	*BatchingEventInterval*	
	BatchingTimeInterval	
	SlidingEventInterval	
	SlidingTimeInterval	
Actions	*Email*	
	Twitter	

5.3. Editor Gráfico

En esta sección se describe el método de desarrollo que se ha seguido para construir el editor gráfico de patrones de eventos. Este método es similar al empleado para construir el editor de dominios CEP (véase la Sección 4.3), al que se le ha añadido un paso más que se encarga de la transformación de los modelos de patrones a código. Por consiguiente, el método se compone de los siguientes 6 pasos:

1. **Creación del metamodelo de dominios CEP**: consiste en la implementación del metamodelo que define el DSML de patrones de eventos propuesto en la Sección 5.2.1. Para ello, el metamodelo se expresa textualmente con Emfatic y se construye con el lenguaje de metamodelo Ecore.

2. **Generación del editor gráfico por defecto**: a partir del metamodelo implementado en el paso anterior y utilizando GMF y EuGENia, se genera automáticamente una versión inicial del editor gráfico.

3. **Personalización del editor generado**: dada las limitaciones, en cuanto a funcionalidad y usabilidad, que presenta el editor GMF generado automáticamente en el paso 2, en este paso se personaliza el editor modificando su paleta de herramientas, así como añadiendo un menú de opciones para que el usuario final pueda llevar a cabo el modelado de patrones de eventos de forma intuitiva. Asimismo, se introducen modificaciones en Java para que el editor pueda reconfigurarse automáticamente a partir de distintos modelos de dominio CEP.

4. **Implementación de las reglas de validación**: se implementan las reglas que permitirán validar si un patrón de eventos modelado está bien formado. Para ello, se emplea el lenguaje EVL del proyecto Epsilon.

5. **Implementación de las transformaciones de modelo a código**: se construyen las reglas que permitirán transformar un modelo de patrón de eventos al código que lo implementa. Para lo cual, se utiliza el lenguaje EGL de Epsilon.

6. **Creación de una aplicación Eclipse RCP**: se crea una aplicación Eclipse RCP que integra todos los *plugins* obtenidos en los pasos anteriores, de forma que esta aplicación —el editor de patrones de eventos— pueda ser ejecutada fuera del IDE Eclipse y en múltiples plataformas.

Seguidamente, se describe pormenorizadamente cada uno de estos pasos.

5.3.1. Creación del Metamodelo de Patrones de Eventos

El primer paso consiste en la implementación del metamodelo de patrones de eventos (véase la Sección 5.2.1). Para cumplir este propósito, se ha utilizado el lenguaje Emfatic, añadiendo a la especificación del metamodelo anotaciones GMF. Estas anotaciones facilitarán posteriormente la generación del editor gráfico, ya que determinan qué elementos del metamodelo se podrán modelar con el editor y cuál será su representación gráfica.

El Listado 5.1 muestra un extracto de la implementación del metamodelo utilizando Emfatic. En la Sección 4.3.1 ya se han explicado algunas de las anotaciones GMF más relevantes. En cuanto a otras anotaciones utilizadas en este metamodelo de patrones caben destacar:

- `source.constraint`: restricción OCL que debe ser comprobada por el editor gráfico al crear un enlace (*Link*). Como puede observarse en la clase *Link*, se han implementado con OCL todas aquellas restricciones que eviten crear enlaces incorrectos entre nodos, atendiendo al metamodelo de patrones de eventos definido en la Sección 5.2.1. Por ejemplo, `self <> oppositeEnd` evitará que el usuario cree un enlace desde un nodo a sí mismo —`self` es el origen del enlace y `oppositeEnd` su destino.

- `figure`: el tipo de figura que representará el nodo. La gran mayoría de los operadores y algunos operandos tienen asociados una figura de tipo SVG (*Scalable Vector Graphics*), proporcionando una mayor calidad y escalabilidad de las imágenes utilizadas para representarlos.

- `label.pattern`: cuando la etiqueta de un nodo está formada por más de un atributo, deberá especificarse el formato que debe utilizarse para mostrar estos valores en la etiqueta. Para ello, se utilizará *Java Message Format*. Este patrón Java será usado tanto para editar la etiqueta como para visualizarla. En el caso de que se desee especificar formatos distintos para editar y para visualizar deberán utilizarse `label.edit.pattern` y `label.view.pattern`, respectivamente.

Listado 5.1: Extracto de la definición del metamodelo de patrones de eventos utilizando la notación textual Emfatic para metamodelos basados en EMF.

```
@namespace(uri="www.uca.es/modeling/cep/eventpattern",prefix="eventpattern")
package eventpattern;

@gmf.diagram(model.extension="pattern", diagram.extension="pattern_diagram")
class CEPEventPattern {
   attr String patternName;
   attr String patternDescription;
   attr String domainName;
   attr Date patternCreationDate;
   attr boolean patternDeployed;
   val Link[*] links;
   val EventPatternElement[*] eventPatternElements;
   val ComplexEvent[0..1] complexEvent;
   val Action[*] actions;
}

@gmf.link(label="order", source="operand", target="operator",
   source.constraint="self <> oppositeEnd and (
      (self.oclIsKindOf(ComplexEvent) and oppositeEnd.oclIsKindOf(Action))
         or ((self.oclIsKindOf(Value) or self.oclIsKindOf(EventProperty) or
         self.oclIsKindOf(AggregationOperator) or self.oclIsKindOf(
         ArithmeticOperator)) and oppositeEnd.oclIsKindOf(
         ComplexEventProperty))  [...] )",
   target.constraint="self <> oppositeEnd", source.decoration="none", target
      .decoration="arrow", color="110,110,110", tool.name="Link", tool.
      description="Add a link")
class Link {
   attr int order;
   ref Operand#outboundLink operand;
   ref Operator#inboundLink operator;
}

abstract class Operator {
   ref Link[*]#operator inboundLink;
}

abstract class Operand {
   ref Link[*]#operand outboundLink;
}

@gmf.node(figure="rectangle", label="typeName", border.color="110,110,110",
   label.view.pattern="New Complex Event: {0}", label.readOnly="true", tool.
   name="New Complex Event", tool.description="Add a new complex event")
class ComplexEvent extends Operand {
   attr String typeName;
   attr String imagePath;
   @gmf.compartment(layout="list")
   val ComplexEventProperty[*] complexEventProperties;
}
```

5.3.2. Generación del Editor Gráfico por Defecto

A partir del metamodelo descrito utilizando Emfatic junto con las anotaciones GMF y utilizando EuGENia se han llevado a cabo automáticamente las siguientes acciones, de forma análoga a como se ha hecho para el editor de dominios CEP:

1. La generación del metamodelo en formato Ecore a partir del definido con Emfatic.
2. La generación del modelo *genmodel* desde el modelo Ecore. El modelo *genmodel* contendrá el código generado asociado al metamodelo Ecore.
3. La generación del código EMF a partir del modelo *genmodel*. Esto generará tres proyectos: *es.uca.modeling.cep.eventpattern.edit* —contendrá el código requerido para poder manipular los modelos de patrones de eventos—, *es.uca.modeling.cep.eventpattern.editor* —contendrá un *plugin* que proporciona un editor de modelos EMF basado en árboles— y *es.uca.modeling.cep.eventpattern.tests* —con el código necesario para realizar pruebas unitarias. Además, el proyecto principal de partida, *es.uca.modeling.cep.eventpattern*, se ha convertido en un proyecto Java con el código correspondiente a cada una de las clases del metamodelo.
4. La generación de los modelos *gmfgraph*, *gmftool* y *gmfmap*.
5. La generación del modelo *gmfgen*, esto es, el modelo de generación de código para GMF.
6. La generación del código correspondiente al editor GMF de patrones de eventos, denominado *es.uca.modeling.cep.eventpattern.diagram*, a partir del modelo *gmfgen*.

Tras la finalización de estas acciones, se ha obtenido un editor GMF generado automáticamente por EuGENia que puede ejecutarse dentro del IDE Eclipse.

5.3.3. Personalización del Editor Generado

Como ya se ha mencionado con anterioridad, a pesar de que la herramienta EuGENia facilita la implementación del editor GMF para manejar modelos de patrones de eventos, el editor generado en el paso anterior presenta limitaciones desde el punto de vista de la funcionalidad y la usabilidad, sobre todo, teniendo en cuenta el tipo de usuario que lo usará. Por este motivo, se ha personalizado tanto la paleta de herramientas como el menú de opciones del editor, tal y como se describirá en las siguientes subsecciones.

En la Figura 5.6 puede observarse el aspecto del editor gráfico de patrones de eventos, tras su personalización, en el que puede distinguirse sus partes principales: la paleta de herramientas, el área de trabajo, el menú de patrones de eventos y la vista de propiedades. Debe tenerse en cuenta que si el usuario final intenta utilizar alguna herramienta de forma incorrecta el editor se lo impedirá, como puede ser el caso, por ejemplo, del intento de enlazar un operador de patrón *Every* con la acción *Email*, lo cual no está permitido según las restricciones OCL descritas en `source.constraint` del metamodelo (veáse la Sección 5.3.1).

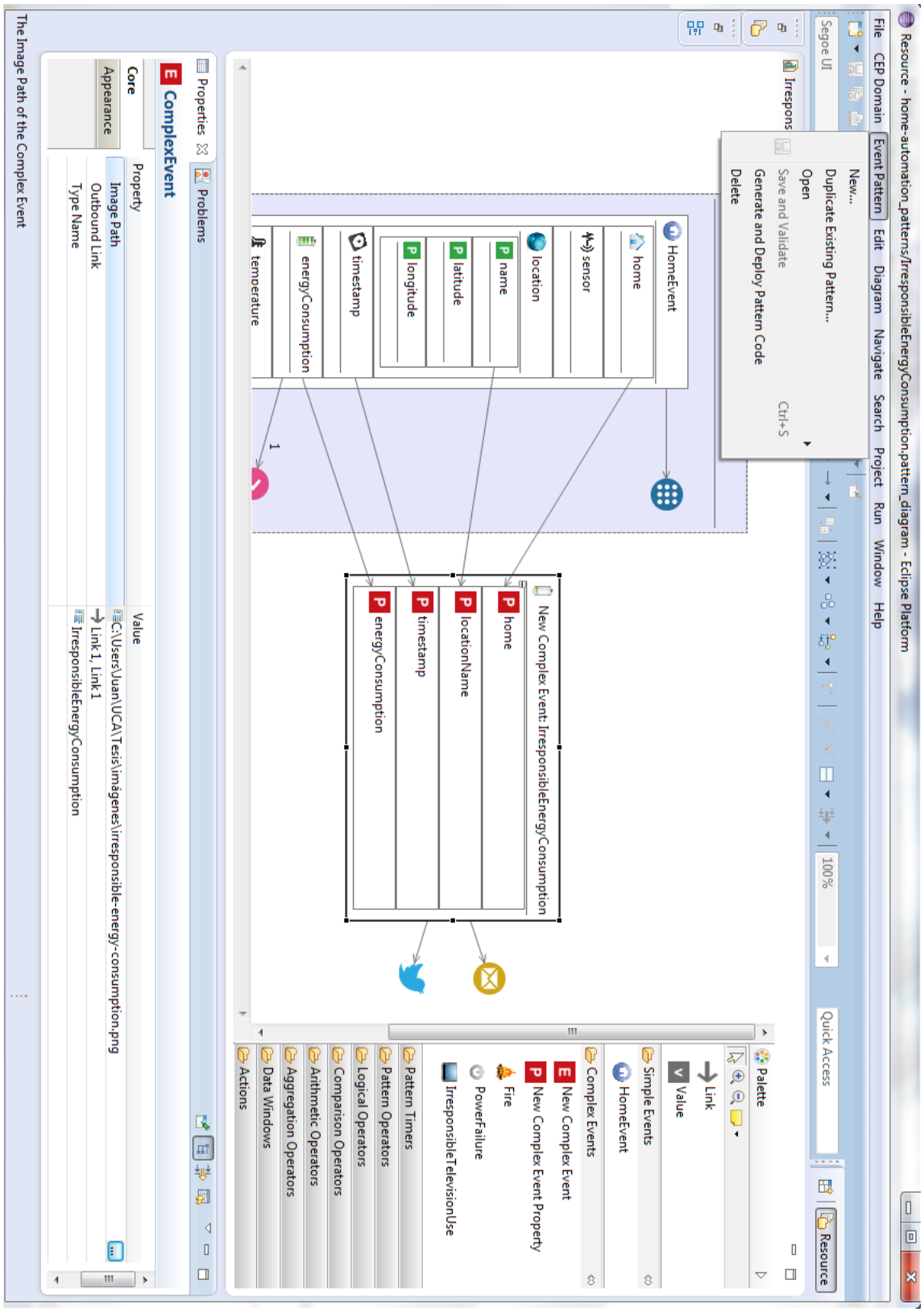

Figura 5.6: Editor gráfico construido para el modelado de patrones de eventos y su transformación a código.

Paleta de Herramientas

La personalización de herramientas del editor se ha realizado en dos partes: una automática, usando una transformación de pulido (*polishing transformation*), y otra manual, modificando el código Java del editor GMF generado.

La personalización automática de la paleta se ha llevado a cabo gracias a la implementación de una transformación de pulido en EOL, el lenguaje principal de Epsilon que es imperativo y combina el estilo procedimental de JavaScript con las capacidades de consulta de OCL. De esta forma, durante la generación del editor GMF por parte de la herramienta EuGENia, esta última se ha encargado, mediante dicha transformación, de realizar todos los cambios pertinentes en los tres modelos GMF: *gmfgraph*, *gmftool* y *gmfmap*. Concretamente, en esta transformación se ha determinado que las herramientas de la paleta deben agruparse en las siguientes diez categorías:

- *Simple Events*: esta categoría agrupa a todos los tipos de eventos simples que existan para un dominio CEP en particular, existiendo una herramienta por cada tipo de evento simple. El nombre de cada una de estas herramientas coincidirá con el tipo de evento que represente y se le asignará, por defecto, un icono verde con la letra *E*, salvo que el usuario final haya sustituido este icono por otra imagen cuya ruta estará especificada en el atributo *imagePath*. Esta funcionalidad extra implementada dota al editor de mayor usabilidad en tanto en cuanto los tipos de eventos simples pueden identificarse mediante imágenes con las que el propio usuario esté familiarizado.

- *Complex Events*: esta categoría está formada por todos los tipos de eventos complejos que se definan durante el diseño gráfico de patrones de eventos para un dominio CEP en concreto. Para proceder a la definición del tipo de evento complejo que detectará el patrón de eventos en cuestión se utilizará tanto la herramienta *New Complex Event* como *New Complex Event Property*, proporcionadas por defecto en esta categoría. Así pues, esta categoría estará formada por dichas herramientas, además de una herramienta adicional por cada tipo de evento complejo definido en los patrones modelados. El nombre de cada una de estas herramientas coincidirá con el tipo de evento complejo que represente y se le asignará por defecto un icono rojo con la letra *E*, salvo que el usuario final haya sustituido este icono por otra imagen cuya ruta estará especificada en el atributo *imagePath*. Esta funcionalidad también proporciona al editor una mayor usabilidad en tanto en cuanto los tipos de eventos complejos pueden identificarse a través de imágenes con las que el propio usuario esté familiarizado.

- *Pattern Timers*: agrupa a todos los operandos de patrón relacionados con temporización, en concreto, *Time Interval*, *Time Schedule* y *Within Timer*.

- *Pattern Operators*: contiene todos los operadores de patrón definidos en el metamodelo (véase la Sección 5.2.1), excepto los operadores *And*, *Or* y *Not* que se incluyen en la categoría *Logical Operators*.

- *Logical Operators*: agrupa a los operadores lógicos propuestos en el metamodelo que, a su vez, son operadores de patrón.
- *Comparison Operators*: contiene todos los operadores de comparación descritos en el metamodelo.
- *Arithmetic Operators*: agrupa a todos los operadores aritméticos definidos en el metamodelo.
- *Aggregation Operators*: contiene todas las funciones de agregación descritas en el metamodelo.
- *Data Windows*: agrupa a todas las ventanas de datos detalladas en el metamodelo.
- *Actions*: contiene las acciones que pueden llevarse a cabo para notificar que un evento complejo ha sido detectado en el sistema.

Es importante destacar que las herramientas *Link* y *Value* no se clasifican dentro de ninguna categoría. Estas se han situado en las primeras posiciones de la paleta del editor, debido a que son las herramientas más utilizadas a la hora de diseñar los patrones de eventos.

Por otra parte, se ha realizado un gran esfuerzo para hacer posible que las herramientas de las categorías de eventos simples (*Simple Events*) y eventos complejos (*Complex Events*) puedan modificarse y gestionarse en tiempo de ejecución. Conviene señalar que lo habitual en editores GMF es que la paleta solo se modifique en tiempo de diseño. Para lograrlo, se han introducido modificaciones en Java sobre el propio código del editor GMF generado.

En cuanto a las herramientas de la categoría de eventos simples, estas dependerán del dominio para el que se haya reconfigurado el editor de patrones de eventos mediante la creación, importación o autodetección de un dominio CEP. Por tanto, si el editor se ha reconfigurado al ámbito de la domótica, como es el caso del editor mostrado en la Figura 5.6, la herramienta *HomeEvent* deberá añadirse a dicha categoría en tiempo de ejecución. Al usar esta herramienta sobre el área de trabajo del editor, aparecerá automáticamente el evento modelado conteniendo todas sus propiedades de eventos.

Del mismo modo, las herramientas de la categoría de eventos complejos también se gestionan en tiempo de ejecución, puesto que dependerán de los patrones de eventos que se diseñen en cada caso. Por consiguiente, si el editor se ha reconfigurado al ámbito de la domótica, se creará automáticamente una herramienta por cada patrón de eventos diseñado y validado. Como puede observarse en la Figura 5.6, se han creado las tres herramientas correspondientes a los patrones de eventos modelados sobre domótica: *Fire*, *PowerFailure* e *IrresponsibleTelevisionUse*. Una descripción detallada de estos patrones puede encontrarse en el Capítulo 7.

Menú de Opciones

El editor de patrones de eventos dispone de la barra de menús disponible en toda aplicación Eclipse. Además de los menús por defecto, se ha añadido el mismo menú *CEP*

Domain del editor de dominios CEP (véase la Sección 4.3.3), así como un nuevo menú específico de este editor denominado *Event Pattern*. Concretamente, las opciones de este nuevo menú son las siguientes:

- *New*: esta opción permite crear un nuevo patrón de eventos, facilitando al usuario final la creación del modelo que represente dicho patrón. Previo a la creación de un patrón de eventos, es requisito indispensable que el editor haya sido reconfigurado con un dominio CEP. Este dominio podrá crearse desde cero, detectarse automáticamente, o bien importarse a partir del archivo *nombre_domain.zip* obtenido al exportar un dominio desde el editor gráfico de dominios CEP. Para más información sobre estas funcionalidades de dominio CEP, consúltese la Sección 4.3.3.

- *Duplicate Existing Pattern*: esta opción facilita la creación de un patrón de eventos a partir de otro ya existente. De este modo, se creará una copia del modelo de patrón de evento activo en ese momento; por lo cual, el usuario final solo tendrá que modificar gráficamente las diferencias que deban existir con respecto al patrón original.

- *Open*: esta opción permite abrir uno de los patrones de eventos que hayan sido definidos previamente.

- *Save and Validate*: esta opción se encargará de guardar el modelo de patrón de eventos que se encuentre activo, así como de validarlo. En caso de que el modelo no esté bien formado, se indicarán cuáles son los problemas encontrados, además de indicarse gráficamente con un icono de error en los elementos implicados del modelo.

- *Generate and Deploy Pattern Code*: esta opción transformará el modelo de patrón de eventos validado al código que lo implementa. Por un lado, se obtendrá el código del patrón en sí cuya sintaxis dependerá del EPL proporcionado por el motor CEP que se desee utilizar. Por otro lado, en caso de que se haya definido alguna acción a llevar a cabo en una SOA 2.0 si se detecta el patrón, entonces se obtendrá también el código correspondiente que ha de ser insertado en el ESB.

- *Delete*: esta opción eliminará uno de los patrones de eventos previamente descritos.

Por otro lado, además de las opciones *Import CEP Domain* y *Export CEP Domain* del editor gráfico de dominios, se han añadido dos opciones más al menú *File* de la barra de menús:

- *Import Event Patterns*: esta opción permitirá importar los patrones de eventos que hayan sido definidos previamente con este editor para un dominio CEP. Para ello, el usuario deberá indicar el archivo a importar cuya nomenclatura seguirá el siguiente formato: *nombre_pattern.zip.*

- *Export Event Patterns*: esta opción se utilizará para exportar los patrones de eventos que hayan sido modelados para un dominio CEP, siempre y cuando los modelos hayan sido validados. Si el usuario ha utilizado imágenes concretas para los tipos

de eventos y las propiedades, entonces dichas imágenes también serán exportadas a tamaño 20 x 20 píxeles. Así pues, se obtendrá un archivo cuya nomenclatura seguirá el siguiente formato: *nombre_pattern.zip*. Este archivo contendrá los modelos de los patrones de eventos (modelos EMF) y sus ficheros de diagrama (diagramas) junto con las imágenes a las que hagan referencia dichos modelos de patrón, así como el modelo de dominio CEP (modelo EMF) para el que se ha definido estos patrones, su fichero de diagrama (diagrama) y las imágenes a las que haga referencia este modelo de dominio.

5.3.4. Implementación de las Reglas de Validación

Las restricciones del metamodelo de patrones de eventos definidas en la Sección 5.2.1 se han implementado con EVL, del mismo modo que se ha hecho con las restricciones del metamodelo de dominios CEP, puesto que, como se ha mencionado en la Sección 4.3.4, OCL presenta algunas limitaciones con respecto a EVL.

5.3.5. Implementación de las Transformaciones de Modelo a Código

Esta es una de las etapas más importantes de la construcción del editor de patrones de eventos, puesto que la finalidad última del editor es precisamente la transformación de los modelos diseñados por los usuarios no tecnólogos a código que pueda ejecutarse en el software requerido para detectar las situaciones críticas o relevantes en tiempo real.

Como se ha comentado previamente, el software seleccionado para detectar las situaciones se compone del motor Esper y del ESB Mule. Por consiguiente, se ha creado un módulo para la transformación de las condiciones de los patrones de eventos modelados a código EPL de Esper que se añadirá automáticamente en el motor Esper, y otro módulo para la transformación de las acciones que se hayan asociado con los patrones al código XML que se ejecutará automáticamente en el ESB Mule; ambos códigos se desplegarán en tiempo de ejecución.

Para llevar a buen término esta tarea, se ha utilizado EGL, el lenguaje de transformación de modelo a texto proporcionado por Epsilon basado en plantillas. En el Listado 5.2 se presenta un pequeño extracto de la plantilla EGL implementada para transformar los modelos de patrones de eventos a código EPL de Esper.

Es importante destacar que, aunque se haya optado por el uso de Esper y Mule, en caso de que se desee utilizar un motor CEP y un ESB diferentes podrá seguir usándose el mismo editor siempre que se creen otras dos plantillas EGL: una plantilla que transforme los mismos modelos de patrones de eventos de partida al EPL específico proporcionado por el motor seleccionado, y otra plantilla que transforme dichos modelos al lenguaje proporcionado por el ESB escogido para ejecutar las acciones de los patrones.

Listado 5.2: Extracto de las reglas implementadas con EGL para transformar los modelos de patrón de eventos a código EPL de Esper.

```
@Name('[%=CEPEventPattern.allInstances.at(0).patternName%]')
[%

createGlobalVariables();

var insertIntoExpression : String;
var selectExpression : String;
var fromPatternExpression : String;
var fromExpression : String;
var whereExpression : String;

[...]

// INSERT INTO expression
var ce : ComplexEvent = ComplexEvent.allInstances().at(0); // A unique
    complex event must exist in the model

if (ce.typeName <> '') {
    insertIntoExpression = "insert into " + ce.typeName;
}

[...]

// Print EPL code
if (insertIntoExpression <> null) {
%][%=insertIntoExpression%]
[% }

if (selectExpression <> null) {
%][%=selectExpression%][% }

if (fromPatternExpression <> null) {
%][%=fromPatternExpression%]
[% }

if (fromExpression <> null) {
%][%=fromExpression%]
[% }

if (whereExpression <> null) {
%][%=whereExpression%]
[% }

[...]

%]
```

5.3.6. Creación de una Aplicación Eclipse RCP

Tras la finalización de los pasos anteriores, se han obtenido los *plugins* que componen la aplicación Eclipse del editor gráfico de patrones de eventos. Sin embargo, esta aplicación requiere ser ejecutada desde dentro del propio IDE Eclipse, lo que impediría que el editor estuviese al alcance de cualquier usuario no informático.

Con el propósito de ofrecer un editor gráfico que pueda ser ejecutado fuera de Eclipse, así como en múltiples plataformas —Windows, Linux y Mac, entre otras—, se ha creado una aplicación Eclipse RCP, del mismo modo que se ha creado para el editor de dominios CEP (véase la Sección 4.3.5).

5.4. Modelado de Patrones de Eventos con el Editor

En el Capítulo 3 se ha propuesto un enfoque dirigido por modelos para el procesamiento de eventos complejos en SOA 2.0. Gracias al editor gráfico de modelado de patrones de eventos presentado en este capítulo, el usuario final ya podrá abordar la ejecución de las fases implicadas durante dicho modelado (véase la Figura 5.7):

- *Definición de modelos de patrón de eventos*: el usuario final será el responsable de definir gráficamente los patrones de eventos para un dominio CEP en particular, como puede ser la salud, la bolsa, la domótica, la seguridad informática, entre otros. Los modelos de patrón de eventos obtenidos serán conformes al metamodelo descrito en la Sección 5.2.1. Para llevar a cabo esta primera fase, será indispensable que el editor de patrones haya sido reconfigurado a priori con un dominio CEP. En caso contrario, el usuario deberá seleccionar una de las siguientes opciones del menú del editor: crear un nuevo dominio CEP desde cero (*CEP Domain > New*), autodetectar el dominio (*CEP Domain > Auto-detect Domain*) o importar el dominio (*File > Import CEP Domain*). Una vez reconfigurado el editor, el usuario podrá crear un nuevo patrón desde cero (*Event Pattern > New*) o como copia de otro (*Event Pattern > Duplicate Existing Pattern*).

- *Validación y almacenamiento de modelos de patrón de eventos*: una vez que los patrones de eventos hayan sido modelados, estos serán validados comprobándose que se cumplen las restricciones definidas en la Sección 5.2.1. Si algún modelo es incorrecto, el editor mostrará los errores que tendrán que solucionarse antes de poder continuar con la siguiente fase. Además, serán guardados en el sistema y podrán exportarse e importarse para reutilizarlos y compartirlos con otros usuarios. En esta fase, deberá usarse la opción de guardar y validar el modelo del patrón activo (*Event Pattern > Save and Validate*) y, en el caso de que se desee importar o exportar los patrones modelados, también deberían usarse las opciones *File > Import Event Patterns* o *File > Export Event Patterns*, respectivamente.

- *Transformación de modelos de patrón de eventos a código*: los modelos de patrón de eventos serán automáticamente transformados tanto al código del patrón correspon-

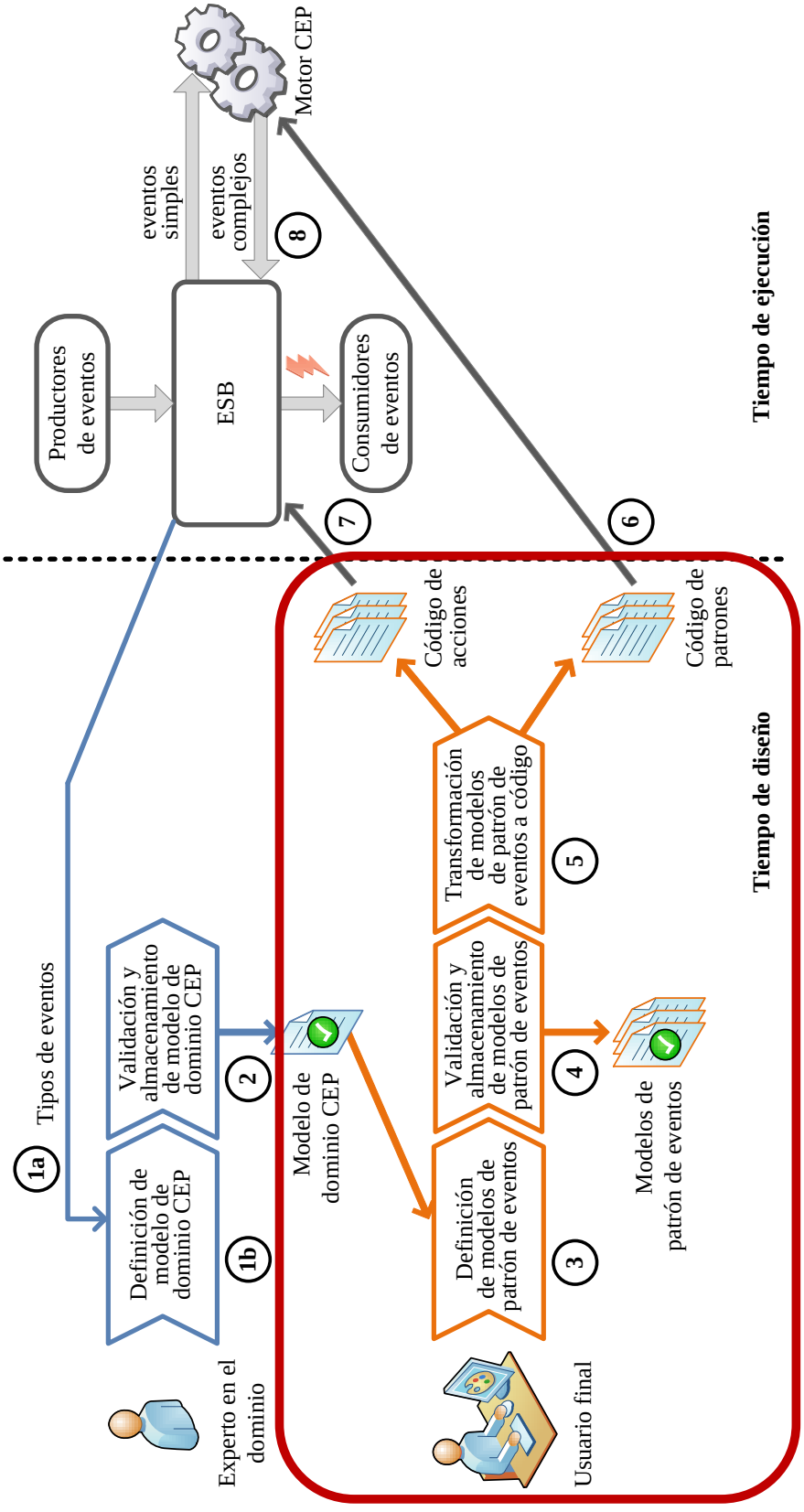

Figura 5.7: Enfoque dirigido por modelos para CEP en SOA 2.0: modelado y generación de código de patrones de eventos.

diente a las condiciones que deben cumplirse para detectar las situaciones críticas dentro del motor CEP, como al código de las acciones a llevar a cabo en el ESB una vez se hayan detectado dichas situaciones. Para ello, el usuario seleccionará la opción *Event Pattern > Generate and Deploy Pattern Code*, debiendo indicar, si todavía no lo ha hecho, el directorio donde desea generar el código del patrón y el directorio donde generar el código de las acciones asociadas al patrón. Posteriormente, los ficheros con extensión *.epl* y *.action* generados en estos directorios serán consumidos automáticamente por el ESB, añadiéndose el patrón EPL en el motor y ejecutándose las acciones en el ESB.

Conviene destacar que en el Capítulo 7 se presentan dos casos de estudio en los que se modelan patrones de eventos sobre domótica y seguridad en redes haciendo uso de este editor gráfico de patrones.

5.5. Conclusiones

En este capítulo se ha presentado un DSML para unificar la descripción de patrones de eventos, representándolos como modelos, facilitando así al usuario final la definición de estos patrones sin necesidad de conocer detalles específicos sobre ningún EPL ni sobre ningún lenguaje requerido para implementar las acciones asociadas a estos patrones. Concretamente, se ha propuesto tanto un metamodelo de patrones de eventos junto con las restricciones que permiten validar los modelos conformes al metamodelo, como su notación gráfica.

Asimismo, se ha construido un editor GMF con el propósito de facilitar a los usuarios finales un entorno amigable e intuitivo con el que podrán definir gráficamente patrones de eventos sobre un dominio donde pueda aplicarse CEP, y sin necesidad de conocer ningún EPL en particular. Además, este editor validará los patrones modelados, comprobando que sean conformes a su metamodelo, generará el código que los implementa, y permitirá importarlos y exportarlos.

Al poder reconfigurarse el editor mediante un modelo de dominio CEP, el usuario dispondrá en todo momento de una interfaz gráfica común adaptada al contexto específico para el que desee definir los patrones de eventos, modificándose dinámicamente la paleta de herramientas dependiendo del dominio en cuestión. Por tanto, el editor pone al alcance de cualquier usuario no tecnólogo la definición de las situaciones críticas o relevantes que necesite detectar en tiempo real.

Capítulo 6

Solución Tecnológica para la Integración del Procesamiento de Eventos Complejos con SOA 2.0

«Nada que dure se contruye en la facilidad.»
(Roger Schutz)

En el Capítulo 3 se ha presentado un enfoque dirigido por modelos para la integración de CEP con una arquitectura orientada a servicios y dirigida por eventos. Para que este enfoque se haga realidad, es requisito indispensable que tanto el editor gráfico para el modelado de dominios CEP, descrito en el Capítulo 4, como el editor gráfico reconfigurable para el modelado y la generación de código de patrones de eventos, definido en el Capítulo 5, sean integrados con la arquitectura orientada a servicios y dirigida por eventos que se propone en este capítulo.

6.1. Motivación

En este capítulo se propone e implementa una SOA 2.0. Tal como se ha explicado en el Capítulo 2, este tipo de arquitectura aúna las ventajas de ambos enfoques: una SOA proporciona una solución eficiente para la implantación de sistemas en los que la modularidad y las comunicaciones entre terceros son un factor clave y una EDA se caracteriza porque las comunicaciones se llevan a cabo por medio de eventos de una forma totalmente desacoplada.

Con vistas a analizar continuamente toda la información que fluye por esta arquitectura ED-SOA, para detectar cuanto antes y de forma automática las situaciones que son críticas para los procesos de negocio, se integra con CEP, tecnología que permite procesar y analizar grandes cantidades de eventos, así como correlacionarlos para detectar y responder en

tiempo real a dichas situaciones. Para ello, se utilizan unos patrones de eventos que inferirán nuevos eventos más complejos y con un mayor significado semántico, que ayudarán a tomar decisiones ante las situaciones acontecidas. Como ya se ha mencionado previamente, esto requerirá el uso de un ESB que actuará como capa de integración, reduciendo el acoplamiento.

6.2. Componentes Integrables en SOA 2.0

En esta sección se describen los principales componentes integrables en la SOA 2.0 que se propone para su integración con CEP. Recuérdese que el elemento principal de este tipo de arquitectura es el ESB.

Como puede verse en la Figura 6.1, los productores de eventos son los componentes de la arquitectura desde los que se obtienen la información que debe ser procesada y correlacionada con la intención de detectar posibles situaciones críticas y/o relevantes en el sistema. Esta información se reconoce como *eventos crudos*, esto es, eventos que tienen el formato y los datos tal cual han sido generados por sus propios productores de eventos. Existen distintos tipos de *productores de eventos*, en esta arquitectura se han considerado algunos de los más relevantes:

- *Generadores de eventos*: aplicaciones diseñadas e implementadas específicamente para generar eventos con un formato determinado para un contexto concreto. Este tipo de productor de eventos es uno de los más frecuentemente usados o implementados cuando no se dispone de sistemas software/hardware más sofisticados para conseguir la información sobre un dominio en particular.

- *Aplicaciones*: programas informáticos que los usuarios utilizan fundamentalmente para gestionar información: insertar, eliminar, actualizar, modificar y consultar datos. Estas aplicaciones pueden ser de distintos tipos, entre los que caben destacar en esta arquitectura propuesta: *aplicaciones web* —los usuarios pueden utilizarlas accediendo a un servidor web a través de un navegador web—, *aplicaciones móviles* —específicamente desarrolladas para su ejecución en teléfonos inteligentes, tabletas y otros dispositivos móviles— y *aplicaciones legacy*—aplicaciones desfasadas en una empresa que, aunque estén anticuadas, se deben seguir utilizando porque la empresa así lo desea o bien porque su reemplazo supondría un gran coste.

- *Servicios web*: aplicaciones modulares que pueden invocarse a través de Internet siguiendo unos estándares establecidos para obtener información [Ort13]. Una de las ventajas principales del uso de servicios web es la interoperabilidad existente entre aplicaciones de software independientemente de las plataformas donde se instalen.

- *Sensores*: dispositivos que monitorizan el entorno donde se encuentran ubicados con el propósito de captar información (temperatura, luminosidad, lluvia, etc.) que posteriormente es transmitida al sistema mediante el controlador que se encuentra integrado en estos dispositivos.

- *Plataformas IoT*: plataformas de Internet de las cosas o IoT (*Internet of Things*) [Atz10, Gub13], normalmente web, que permiten gestionar la información proveniente de sensores localizados a lo largo de todo el planeta, permitiendo que los usuarios y las aplicaciones autorizados puedan acceder a esta información para analizarlos y procesarlos. Estas plataformas son ideales para la obtención de volúmenes masivos de datos heterogéneos proporcionados por una gran cantidad de usuarios, organizaciones y compañías a nivel mundial, sin el requerimiento de grandes inversiones ni en software ni en hardware por parte de los interesados en estos datos.

- *Servicios de redes sociales*: medios de comunicación social cuyo principal objetivo es fomentar las relaciones personales a través de Internet. Estos servicios de redes sociales pueden tratarse como productores de eventos, obteniéndose así una ingente cantidad de información y heterogénea, en tiempo real, sobre lo que publican los usuarios a nivel mundial. Por consiguiente, proporcionan un método veloz y escalable para poner estos datos al servicio de una gran audiencia [Luc12].

Los componentes homólogos a los productores de eventos son los denominados *consumidores de eventos*. Estos son los componentes que reaccionan ante los eventos recibidos desde dicha arquitectura. Al igual que ocurre con los productores de eventos, existe una gran variedad de consumidores de eventos, entre los que caben destacar:

- *Consolas de monitorización*: aplicaciones diseñadas específicamente para avisar a los usuarios de las situaciones críticas y/o relevantes acontecidas en tiempo real. Fundamentalmente, mostrarán los eventos simples y los eventos complejos de una forma textual y/o gráfica, cuya funcionalidad principal es el aviso de las alarmas detectadas. La ventaja del uso de consolas de monitorización gráficas es que proporcionan mayor usabilidad frente a las textuales.

- *Aplicaciones*: aplicaciones, principalmente de tipo web y móviles, que podrán recibir los eventos complejos detectados en el sistema.

- *Servicios web*: estos servicios pueden implementarse con el propósito de facilitar el registro de eventos complejos en sistemas y plataformas heterogéneos.

- *Actuadores*: dispositivos que realizan alguna acción en particular (apagado/encendido, apertura/cierre, etc.) en el lugar donde se encuentran ubicados, tras la detección de algún evento complejo.

- *Plataformas IoT*: plataformas conectadas con una gran cantidad de sensores ubicados por todo el mundo que permitirán el registro de los eventos complejos que se detecten en el sistema.

- *Servicios de redes sociales*: a través de estos servicios se difundirán rápidamente los eventos complejos detectados a una vasta red de usuarios que hayan mostrado previamente interés en estos eventos.

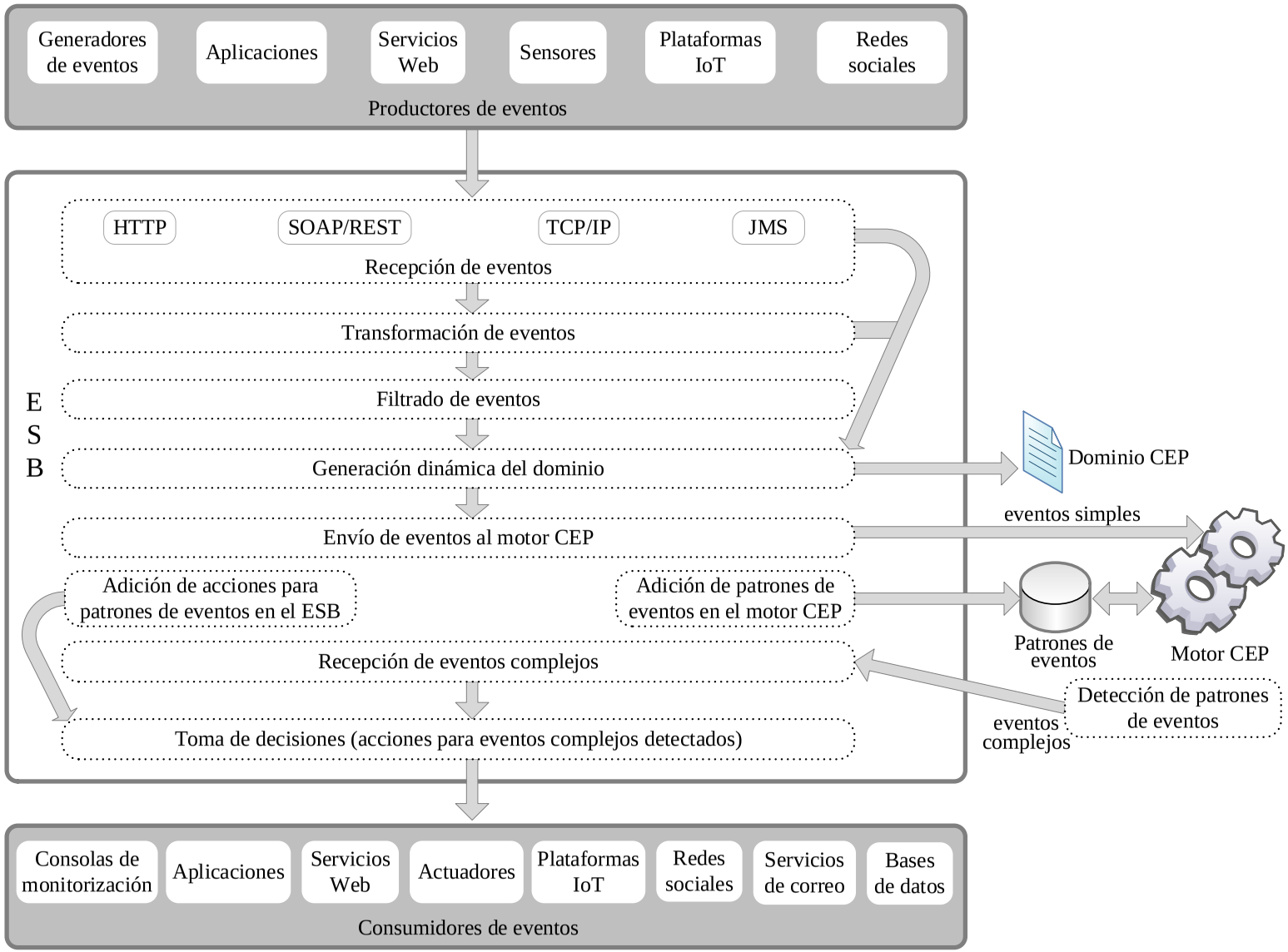

Figura 6.1: Propuesta para la integración de CEP con SOA 2.0.

- *Servicios de correo electrónico*: servicios que permiten la gestión de direcciones de correo electrónico a las que se enviarán los eventos complejos conforme se detecten en el sistema.

- *Bases de datos*: colecciones de datos donde se almacenarán los eventos complejos detectados en el sistema y otros datos de interés como, por ejemplo, la estructura y el formato (metadatos), así como los propios datos de los eventos que fluyan por la arquitectura. Esta arquitectura permite la integración tanto con bases de datos de tipo relacional o SQL como no relaciones o NoSQL (*Not only SQL*) [Cat11, Sto10].

Estos productores y consumidores de eventos se conectan mediante un ESB. La distribución de los eventos por la arquitectura se realiza normalmente de forma asíncrona, es decir, los productores de eventos envían los eventos y, más tarde, los consumidores de eventos los reciben. Además, este mecanismo de distribución de eventos suele ser uno-a-muchos, por lo que tras el envío de un evento este podrá ser recibido por múltiples consumidores de eventos.

Es importante recordar que los productores de eventos y los consumidores de eventos están totalmente desacoplados, por lo tanto, los productores de eventos no dependen de las acciones que deban realizar los consumidores de eventos, y viceversa. Dicho de otro modo, los productores de eventos están estrechamente relacionados con los eventos que producen, pero no tienen ninguna relación directa con los consumidores de eventos, tampoco son conscientes de cuántos consumidores de eventos hay en el sistema ni saben qué acciones llevarán a cabo los consumidores cuando reciban los eventos; asimismo, dichos consumidores reaccionan ante los eventos por sí mismos sin depender de los productores de eventos que los generan.

6.3. Funcionalidades Integradas en SOA 2.0

Las funcionalidades integradas en la SOA 2.0 propuesta son las siguientes:

- *Recepción de eventos*: en esta etapa se reciben los eventos tal cual han sido generados por los productores de eventos, denominados *eventos crudos*. Para cada tipo de estos productores se utilizarán adaptadores que se encargarán de mediar entre protocolos de transportes distintos como, por ejemplo, JMS (*Java Message Service*) para generadores de eventos, HTTP (*HyperText Transfer Protocol*) para plataformas IoT, SOAP/REST (*Simple Object Access Protocol/REpresentational State Transfer*) para servicios web, TCP/IP (*Transmission Control Protocol/Internet Protocol*) para sensores, HTTP/JMS para aplicaciones.

- *Transformación de eventos*: todos los eventos crudos, independientemente de donde provengan, deberán compartir el mismo formato —preferiblemente formato *maps* de Java— para facilitar al motor CEP su procesamiento. En caso contrario, estos eventos serán transformados a un formato común.

- *Filtrado de eventos*: en el caso de que sea necesario, podrán filtrarse los eventos para enviar al motor CEP únicamente aquellos que cumplan alguna condición.

- *Generación dinámica del dominio*: previo al envío de cada evento al motor CEP, se comprobará si su tipo ya es reconocido por el sistema o si se trata de un nuevo tipo de eventos. En este último caso, se inferirá automáticamente tanto el tipo de eventos como los tipos de sus propiedades, incluso si estas se encuentran anidadas, registrándose por consiguiente el nuevo tipo de eventos con sus propiedades en el motor CEP.

- *Envío de eventos al motor CEP*: una vez recibidos los eventos en el ESB, y quizás transformados y/o filtrados, serán enviados e introducidos en los flujos de eventos del motor CEP en tiempo de ejecución.

- *Adición de patrones de eventos en el motor CEP*: el código de los patrones de eventos que se deseen detectar podrá ser añadido al motor CEP en tiempo de ejecución. Estos patrones describirán las condiciones que deben cumplirse para detectar las situaciones relevantes y/o críticas en este sistema.

- *Adición de acciones para patrones de eventos en el ESB*: el código de las acciones que se deseen llevar a cabo para los patrones de eventos podrá ser añadido al ESB en tiempo de ejecución.

- *Detección de patrones de eventos*: en el caso de que se cumplan las condiciones especificadas en uno de los patrones de eventos registrados en el motor CEP, se creará un evento complejo, representando cuál es la situación recién acontecida, que será enviado al ESB.

- *Recepción de eventos complejos*: el ESB recibirá los eventos complejos conforme se vayan creando a partir de la detección de los patrones de eventos.

- *Toma de decisiones (acciones para eventos complejos detectados)*: el ESB se encargará de enviar los eventos complejos a los consumidores de eventos que se hayan suscrito a estos tipos de eventos. Por ejemplo, los eventos complejos podrán ser notificados a los usuarios por correo electrónico o a través de una cuenta Twitter. Además, también podrán almacenarse en un *repositorio de eventos*, una base de datos que se utilizará como el histórico de eventos, facilitándose así el procesamiento retrospectivo, esto es, la realización de consultas sobre eventos pasados. Se recomienda que esta base de datos sea de tipo NoSQL, debido a que son propicias para el almacenamiento en tiempo real y la gestión de grandes cantidades de datos, entre otras ventajas.

Debe tenerse en cuenta que las funcionalidades de *recepción de eventos*, *transformación de eventos* y *filtrado de eventos* son dependientes del escenario donde se aplique esta SOA 2.0, es decir, según el productor de eventos que proporcione los eventos implicará modificaciones en la *recepción de eventos* como, por ejemplo, la utilización de un adaptador JMS o de un

adaptador HTTP; además de otras modificaciones en la *transformación de eventos* y *filtrado de eventos*, utilizándose en este caso un transformador JMS a *Map* o un transformador HTTP a *Map*, respectivamente, e incluso efectuándose su filtrado.

6.4. Implementación de SOA 2.0 y su Integración con CEP

En esta sección se describe la implementación de la SOA 2.0 propuesta y su integración con los editores gráficos de modelado, utilizando el ESB Mule y el motor CEP Esper.

Mule usa los flujos (*flows*) como estructura de control principal para llevar a cabo todo el proceso y gestión de los mensajes que fluyen por este bus. Un flujo Mule se compone normalmente de los siguientes elementos:

- *Endpoints*: permiten que las aplicaciones Mule puedan comunicarse con el *mundo* exterior. Se clasifican en: *inbound* —la aplicación recibirá información del exterior– y *outbound* —la aplicación enviará información al exterior. Algunos de los tipos de *endpoints* son los siguientes: un *endpoint inbound* HTTP ofrece la posibilidad de obtener datos de una fuente de información, a partir de la URL (*Uniform Resource Locator*) que se indique y haciendo uso de este protocolo; un conector *File* hace posible que una aplicación Mule intercambie ficheros con el sistema de ficheros del sistema operativo en cuestión; un VM (*Virtual Machine*) permite establecer comunicaciones internas entre los propios flujos de Mule mediante colas en memoria.

- *Scopes*: proporcionan diferentes formas de combinar o agrupar varios procesadores de mensajes —bloques que permiten filtrar, encaminar o validar mensajes— con el objetivo de mejorar la legibilidad del flujo, implementar procesamiento paralelo y/o crear secuencias de bloques reutilizables. Por ejemplo, un *composite source* posibilita que dos o más fuentes de mensajes pueden agruparse con la intención de aceptar mensajes provenientes de distintas fuentes externas. Por otro lado, un *flow reference* facilita el envío de mensajes desde un flujo determinado a otro.

- *Componentes*: se encargan de recibir mensajes y devolverlos una vez que hayan sido procesados, añadiendo, por tanto, funcionalidad a un flujo en particular, como puede ser la impresión por consola del contenido de los mensajes que fluyan por él (*Echo* o *Logger*). Asimismo, también ofrecen la posibilidad de que el propio desarrollador implemente cómo procesar estos mensajes, mediante el uso de distintos lenguajes de programación, tales como Java, JavaScript, Python. Como se detallará más adelante, en la solución tecnológica propuesta se han creado algunos componentes propios usando Java.

- *Transformadores*: pueden convertir los mensajes de entrada en un formato de datos consumible por otros procesadores de mensajes del flujo. Mule proporciona por defecto algunos transformadores como, por ejemplo, *File to String*, para convertir todo

el contenido de un fichero en una cadena de texto. Además, se pueden implementar transformaciones personalizadas, tal y como se ha llevado a cabo en la solución tecnológica para transformar, por ejemplo, mensajes en formato JSON (*JavaScript Object Notation*) [JSO14] a formato *map* de Java.

- *Filtros*: determinan si un mensaje puede continuar a través del flujo de la aplicación, o si debe rechazarse.
- *Controles de flujos*: especifican cómo los mensajes serán encaminados hacia distintos procesadores de mensajes dentro de un flujo. También pueden procesar mensajes (agregar o separar su contenido) antes de encaminarlos a otros procesadores de mensajes.
- *Manejadores de errores*: ofrecen varios procedimientos para manejar excepciones bajo ciertas circunstancias.

Con el propósito de implementar dicha SOA 2.0, se ha creado una aplicación Mule con los siguientes 5 flujos:

- *EventReceptionAndManagement-Domain1*: este flujo engloba las funcionalidades previamente descritas de *recepción de eventos*, *transformación de eventos*, y *filtrado de eventos*. Es importante señalar que este es el único flujo que dependerá del escenario concreto al que se desee aplicar la SOA 2.0, lo que implicará su modificación dependiendo de los tipos de productores de eventos con los que se conecte. Para esta tarea, se requerirá tener conocimientos en el manejo de Mule Studio [Mul14], un IDE basado en Eclipse para desarrollar, depurar y desplegar aplicaciones Mule que ofrece un editor gráfico *drag-and-drop* y un editor de código XML.
- *DomainDynamicGenerationAndEventSendingToEsper*: este flujo implementa las funcionalidades de *generación dinámica del dominio* y *envío de eventos al motor CEP*, donde se comprueba si existen nuevos tipos de eventos, registrándose en el motor Esper en caso afirmativo, y además se envían a este motor todos los eventos recibidos en este flujo.
- *EventPatternAdditionToEsper*: este flujo se corresponde con la funcionalidad de *adición de patrones de eventos en el motor CEP*. En particular, se comprobará cada 2 segundos si existe en el directorio determinado por el usuario un nuevo fichero con el código EPL, generado automáticamente por el editor de patrones de eventos. En caso afirmativo, registrará este patrón en el motor Esper.
- *ActionForEventPatternAdditionToMule*: este flujo implementa la funcionalidad de *adición de acciones para patrones de eventos en el ESB*. Concretamente, se comprobará cada 2 segundos si existe en el directorio en cuestión un nuevo fichero con el código XML de las acciones para cada patrón de eventos, generado automáticamente también por el editor de patrones de eventos. En caso afirmativo, se creará un flujo dinámico Mule con estas acciones.

- *ComplexEventReceptionAndDecisionMaking*: este flujo engloba las funcionalidades de *detección de patrones de eventos*, *recepción de eventos complejos* y *toma de decisiones*. Concretamente, recibe los eventos complejos creados por el motor Esper tras la detección de alguno de los patrones de eventos registrados por el flujo *EventPattern-AdditionToEsper* y, además, ejecuta de los flujos dinámicos Mule creados en el flujo *ActionForEventPatternAdditionToMule*, aquel que gestione las acciones que deban llevarse a cabo para dichos eventos complejos. Gracias a la ejecución de dicho flujo dinámico, los eventos complejos serán notificados solo a los consumidores de eventos interesados en recibirlos.

Las Figuras 6.2 y 6.3 muestran estos flujos implementados utilizando Mule Studio. Cabe destacar que, aun utilizando Mule Studio, ha sido necesario la implementación manual tanto de código Java como XML para dotar de todas las funcionalidades descritas a la SOA 2.0 propuesta. Seguidamente, se especificarán los detalles técnicos más relevantes de cada uno de los flujos Mule implementados.

6.4.1. Recepción y Gestión de Eventos

El flujo de recepción y gestión de eventos tendrá al menos un *endpoint* que permitirá obtener en Mule los datos provenientes de un productor de eventos. Opcionalmente, podrá ser conectado a un transformador que se encargará de convertir los eventos obtenidos a un formato adecuado para ser procesado por el motor Esper, preferiblemente *maps* de Java. Una vez que los eventos dispongan del formato adecuado, se podrá usar un filtro que identifique eventos inválidos atendiendo a algún criterio establecido y que rechace su paso al resto del flujo. Finalmente, el flujo terminará con un componente *flow reference*, que se encargará de enviar los eventos que le lleguen al siguiente flujo denominado *DomainDynamicGenerationAndEventSendingToEsper*.

El hecho de haber separado las funcionalidades de *generación dinámica del dominio* y *el envío de eventos al motor CEP* de este flujo, denominado *EventReceptionTransformationEnrichmentAndFiltering-Domain1*, que se encarga de la recepción de eventos así como de la transformación y filtrado de eventos, conlleva que puedan implementarse otros flujos análogos como, por ejemplo, *-Domain2* y *-Domain3*, uno por cada dominio de aplicación, sin que esto implique cambios en el resto de los flujos de la arquitectura propuesta. Los únicos requisitos que se exigen a estos nuevos flujos son que dispongan, al menos, de un *endpoint* que se conectará con un productor de eventos distinto del que se obtendrán los datos, un transformador —en el caso de que los eventos recibidos no estén en formato *maps* de Java— y un componente que referencie al flujo *DomainDynamicGenerationAndEventSendingToEsper*. Esto hará posible que la arquitectura pueda ser extendida y conectada con más tipos de productores de eventos, simplemente añadiendo nuevos flujos Mule sin necesidad de modificar los ya existentes.

En la Figura 6.2 puede observarse que el flujo de recepción y gestión de eventos, denominado *EventReceptionAndManagement-Domain1*, dispone de dos *endpoints* HTTP de entrada —nombrados como *Residential Information* y *Current Cost Bridge*—, un trans-

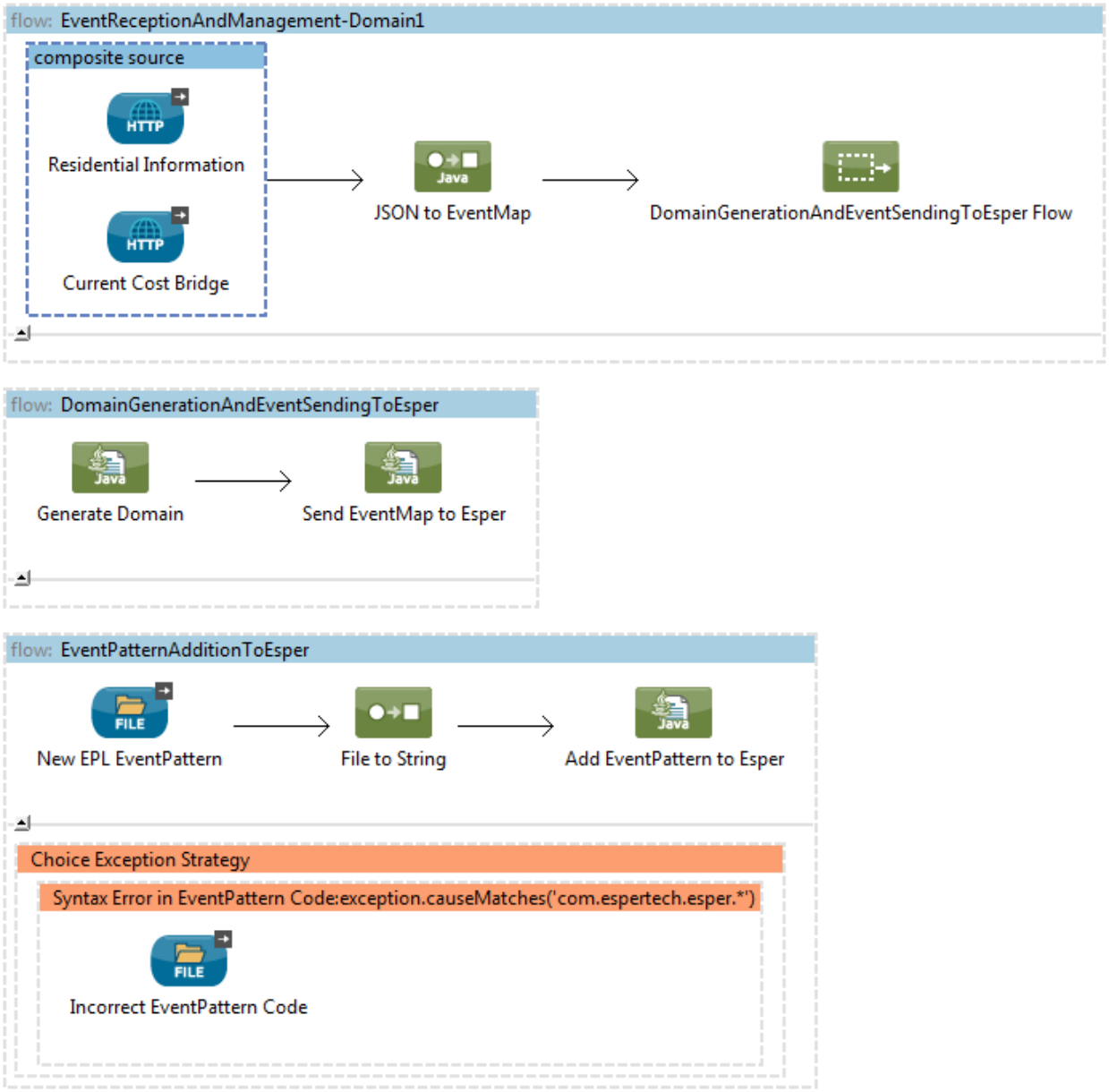

Figura 6.2: Implementación de la SOA 2.0 propuesta (I).

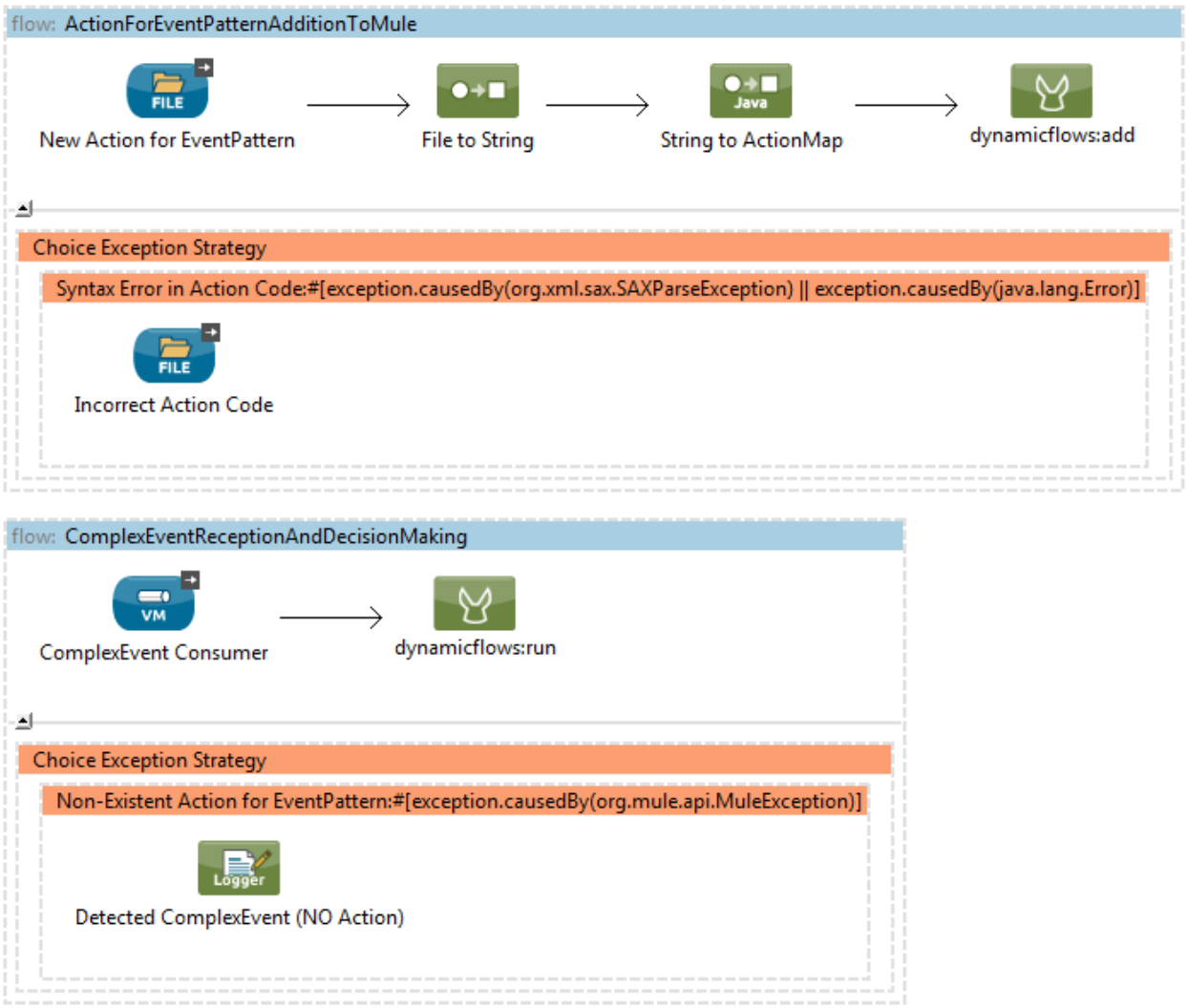

Figura 6.3: Implementación de la SOA 2.0 propuesta (II).

formador de formato JSON a *Map* y, finalmente, un componente de referencia al flujo *DomainDynamicGenerationAndEventSendingToEsper*, por lo tanto, dispone de los elementos mínimos exigidos propuestos para todo flujo de recepción y gestión de eventos. Una descripción más detallada de dichos *endpoints* puede encontrarse en el caso de estudio sobre domótica (véase la Sección 7.1).

Para convertir los datos obtenidos en formato JSON a eventos de tipo *map* de Java, se ha creado un transformador en Java denominado *JSON to EventMap.* Con la finalidad de simplificar la implementación, se ha utilizado Jackson [Jac14], una biblioteca Java multipropósito para procesar datos en formato JSON. Este transformador obtiene el cuerpo del mensaje (*payload*) de tipo *MuleMessage* recibido y, a partir de la información contenida en este *payload*, se creará y devolverá un nuevo *map* de Java, denominado *eventMap* de tipo *Map<String, Object>*, que almacenará todas las propiedades de dicho evento con sus respectivos valores.

Finalmente, el componente de referencia de flujo enviará cada *eventMap* recibido con los datos en el formato apropiado al flujo *DomainDynamicGenerationAndEventSendingToEsper.*

Puede consultarse el Capítulo 7 donde se proponen dos casos de estudio en los cuales se utiliza esta arquitectura; en cada caso de estudio se ha implementado un flujo distinto de recepción y gestión de eventos.

6.4.2. Generación Dinámica del Dominio y Envío de Eventos al Motor CEP

La generación dinámica del dominio es una de las funcionalidades más destacables de la integración de la arquitectura propuesta con los editores gráficos de modelado desarrollados en esta tesis doctoral. Básicamente consiste en el análisis de los eventos que fluyen por la arquitectura con el objeto de obtener automática y dinámicamente el domino CEP, esto es, se inferirá automáticamente tanto el tipo de eventos como los tipos de sus propiedades, incluso si estas se encuentran anidadas.

Este flujo Mule cuya misión principal es la *generación dinámica del dominio* y el *envío de eventos al motor CEP* se ha implementado mediante un componente Java, denominado *GenerateDomain*, donde se comprueba si existen nuevos tipos de eventos en el ESB, registrándose en el motor Esper en caso afirmativo, y a través de otro componente Java, denominado *Send EventMap to Esper*, para el envío al motor de todos los eventos recibidos en este flujo.

Además, cada nuevo tipo de evento detectado será serializado generándose, por tanto, una secuencia de *bytes* que será almacenada en un fichero denominado igual que el tipo y con extensión *.eventtype*. Este fichero será posteriormente leído por el editor de dominios CEP (véase el Capítulo 4) para obtener el tipo de evento detectado y proceder a la creación automática de un modelo gráfico de dominio CEP, o su modificación, con este nuevo tipo de evento. Finalmente, el usuario final podrá modificar este modelo de dominio CEP introduciendo, por ejemplo, su descripción textual.

Por otro lado, al componente *Send EventMap to Esper* llegará el mismo evento recibido por el componente *GenerateDomain* que se enviará al motor Esper. El motor será capaz de reconocer el tipo de este evento al haber sido previamente registrado durante la generación del dominio.

6.4.3. Adición del Código de Patrones de Eventos en el Motor CEP

El flujo de adición del código de patrones de eventos en el motor CEP, denominado *EventPatternAdditionToEsper,* facilita la inserción en tiempo de ejecución de nuevos patrones de eventos en el motor Esper, a partir de un fichero por patrón conteniendo el código EPL del patrón que se desee añadir. De esta forma, el registro de nuevos patrones de eventos en el motor no implicará ninguna modificación en los flujos Mule, lo que supone una gran ventaja para la arquitectura propuesta.

El primer elemento de este flujo es un *endpoint* de tipo *File*, denominado *New EPL EventPattern*, que comprobará cada 2 segundos si existe un nuevo fichero con extensión *.epl* en el directorio *new-eventpattern* (véase un ejemplo de este tipo de fichero en el Listado 7.1). En caso afirmativo, el fichero será convertido a tipo *String*, haciendo uso del transformador *File to String*, y enviado al componente *Add EventPattern to Esper*. Este componente añadirá el nuevo patrón de eventos en el motor Esper y le asociará un *listener*. Si en un momento determinado las condiciones de este patrón son satisfechas, entonces el motor creará un evento complejo que será enviado a dicho *listener*, el cual se encargará de enviarlo de vuelta al ESB con el formato adecuado. En concreto, el evento complejo se difundirá a un *endpoint* VM denominado *ComplexEvent Consumer* y localizado en el flujo *ComplexEventReceptionAndDecisionMaking*. Este componente es ideal para realizar comunicaciones internas entre los propios flujos de Mule mediante colas en memoria. Nótese que en el caso de que se produzca alguna excepción de tipo `com.espertech.esper.*`, bien porque la sintaxis del código EPL sea incorrecta o bien por cualquier otro motivo, el patrón no se añadirá al motor y, además, se moverá el fichero *.epl* del directorio *new-eventpattern* al directorio *incorrect-eventpattern*. Sin embargo, si todo el proceso se ha realizado con éxito, el fichero será movido al directorio *deployed-eventpattern.*

6.4.4. Adición del Código de las Acciones en el ESB

El flujo *ActionForEventPatternAdditionToMule* permite registrar en tiempo de ejecución las acciones que deberán ejecutarse automáticamente en el ESB cuando se detecte un patrón en particular. Tal y como está implementada la arquitectura, esto no requerirá ninguna modificación en los flujos Mule ya creados.

El primer elemento de este flujo es un *endpoint* de tipo *File*, denominado *New Action for EventPattern*, que comprobará cada 2 segundos si existe un nuevo fichero con extensión *.action* en el directorio *new-action* (véase un ejemplo de este tipo de fichero en el Listado 7.2). En caso afirmativo, el fichero será convertido a tipo *String* y enviado al componente *String to ActionMap*. Este componente se encarga de crear y devolver un *map* de

tipo `Map<String, Object>`. Finalmente, el componente *dynamicflows:add* registrará en tiempo de ejecución el flujo especificado. Para que este flujo se ejecute será requisito indispensable utilizar el componente *dynamicflows:run*, tal y como se verá en el flujo *ComplexEventReceptionAndDecisionMaking*. Téngase en cuenta que en el caso de que se produzca alguna excepción de tipo `org.xml.sax.SAXParseException` o `java.lang.Error` el nuevo flujo con las acciones no se registrará en el ESB y, además, se moverá el fichero *.action* del directorio *new-action* al directorio *incorrect-action*. Sin embargo, si todo el proceso se ha realizado con éxito, el fichero será movido al directorio *deployed-action.*

6.4.5. Recepción de Eventos Complejos y Toma de Decisiones

Este flujo, denominado *ComplexEventReceptionAndDecisionMaking*, recibirá todos los eventos complejos difundidos por el *listener* y, a continuación, se encargará de tomar decisiones automáticamente, esto es, realizará todas las acciones especificadas previamente para el patrón de eventos que haya creado dicho evento complejo.

En primer lugar, se utiliza un *endpoint* VM, denominado *ComplexEvent Consumer*, donde se recibirá cada evento complejo difundido por el *listener* junto con el nombre del patrón de eventos que lo haya detectado. A continuación, se enviará al componente *dynamicflows:run*. Este componente ejecutará uno de los flujos añadidos dinámicamente por el componente *dynamicflows:add*, situado en el flujo *ActionForEventPatternAdditionToMule*; concretamente, se ejecutará el flujo dinámico que tenga el mismo nombre que el patrón que haya creado el evento complejo recibido en dicho *endpoint.* De esta forma, se consigue que el evento complejo se difunda a todos los consumidores de eventos indicados en dicho flujo dinámico, es decir, se envíe solo a los usuarios/sistemas que estén interesados en detectar en tiempo real estas situaciones críticas y/o relevantes descritas por estos eventos complejos. En el caso de que no se haya añadido ningún flujo dinámico con las acciones a llevar a cabo para un patrón determinado, entonces se lanzará una excepción de tipo `org.mule.api.MuleException`, mostrándose un mensaje de aviso.

6.5. CEP en SOA 2.0 al Alcance del Experto en el Dominio

La solución tecnológica para la integración de CEP con SOA 2.0 propuesta e implementada en el presente capítulo se ha integrado tanto con el editor gráfico para el modelado de dominios CEP como con el editor gráfico para el modelado y la generación de código de patrones de eventos, tal y como se describe en el enfoque dirigido por modelos presentado en el Capítulo 3. En particular, se ha abordado el desarrollo de las fases del enfoque implicadas durante la generación dinámica del dominio, la adición de nuevos patrones de eventos en el motor CEP y de nuevas condiciones a detectar en el ESB, así como la detección de situaciones críticas y su difusión en tiempo de ejecución (véase la Figura 6.4):

- A partir de la información que llegue al ESB proveniente de los productores de

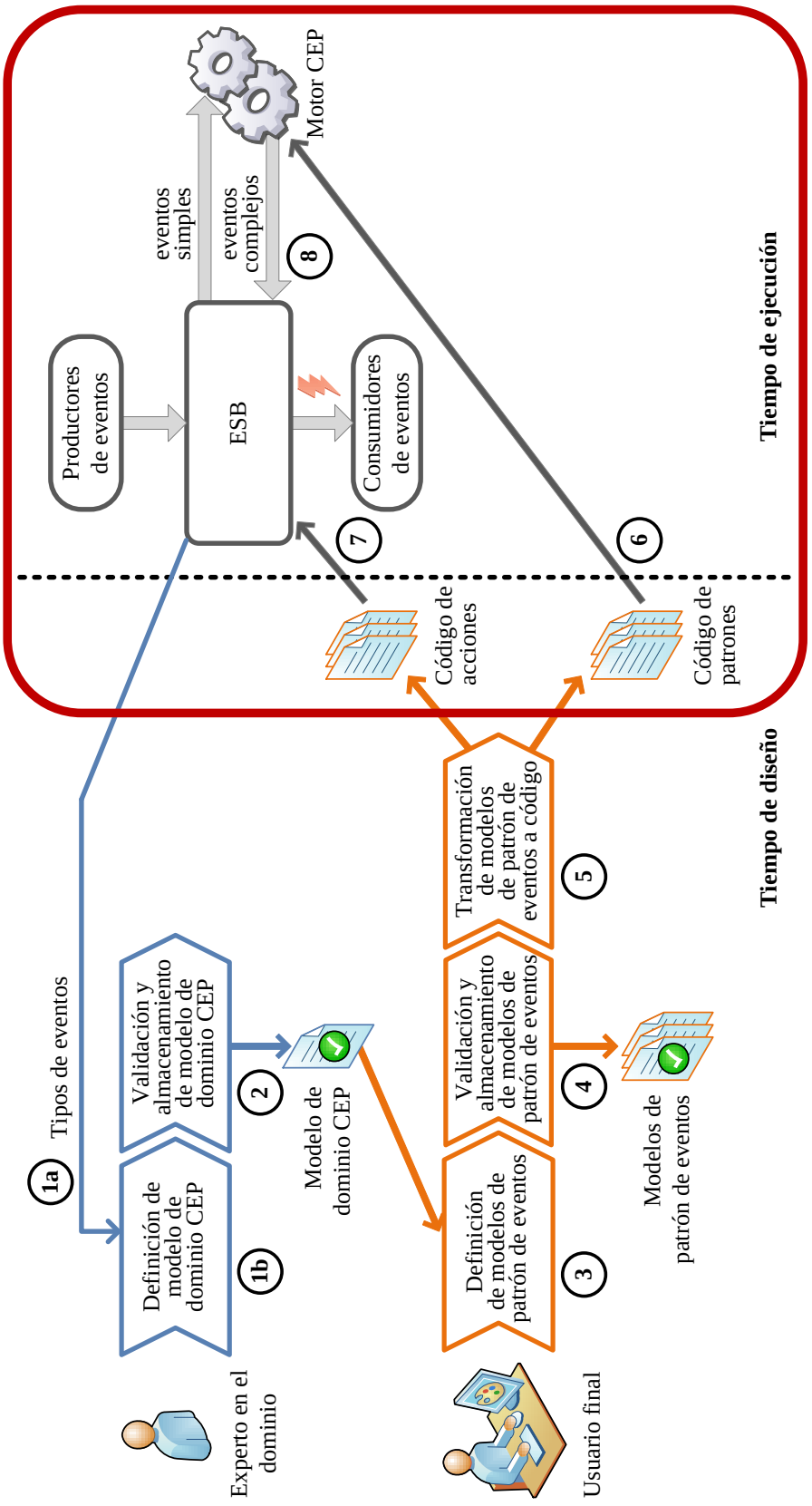

Figura 6.4: Enfoque dirigido por modelos para CEP en SOA 2.0: tiempo de ejecución.

eventos, se generará automáticamente un modelo de dominio con los tipos de eventos y propiedades que fluyan por la arquitectura.

- El código del patrón de eventos generado automáticamente por el editor de patrones será añadido al motor CEP en tiempo de ejecución.
- El código de las acciones generado automáticamente también por el editor de patrones será añadido al ESB en tiempo de ejecución.
- A partir de los eventos simples que lleguen al motor CEP desde el ESB, y los patrones de eventos añadidos en tiempo de ejecución, el motor CEP creará nuevos eventos complejos (alertas o alarmas) conforme se vayan detectando dichos patrones. Estos eventos complejos serán enviados al ESB que se encargará de difundirlos a todos los consumidores de eventos interesados en ellos —según se especifique en las acciones añadidas al ESB por cada patrón de eventos—, avisando al usuario final de las situaciones recién acontecidas.

Así pues, con el propósito de que el usuario final no tenga que implementar a mano el código de los patrones de eventos que necesite detectar en tiempo real, se ha integrado esta arquitectura con el editor gráfico reconfigurable (véase el Capítulo 5). De este modo, los patrones de eventos que sean definidos gráficamente por los usuarios mediante este editor serán transformados a código EPL, creándose un fichero por cada modelo de patrón con el mismo nombre y con extensión *.epl*. Estos ficheros podrán ser alojados automáticamente por el editor en el mismo directorio *new-eventpattern* configurado en este flujo Mule para que sean consumidos y registrados en el motor CEP por el ESB.

Por otro lado, la integración de la arquitectura con el editor gráfico reconfigurable, también facilita que el usuario final no tenga que implementar a mano el código XML de las acciones a llevar a cabo en tiempo real cuando se detecten ciertos patrones de eventos. Por lo cual, las acciones para patrones que sean definidas gráficamente por los usuarios mediante este editor serán transformadas a código XML, creándose un fichero por cada modelo de patrón con el mismo nombre y con extensión *.action*. Estos ficheros podrán ser alojados automáticamente por el editor en el mismo directorio *new-action* configurado en este flujo Mule para que sean consumidos y registrados dinámicamente por el ESB.

Por consiguiente, las principales aportaciones de esta integración son la generación dinámica del dominio de forma gráfica, infiriéndose cuáles son los tipos de eventos y propiedades a partir de los eventos que fluyen por la arquitectura, así como la adición automática y transparente en tiempo de ejecución tanto del código de los patrones de eventos gráficos en el motor CEP como del código de las acciones en el ESB.

6.6. Conclusiones

En este capítulo se ha presentado e implementado una propuesta para la integración de CEP con SOA 2.0 que hace posible la detección de situaciones críticas o relevantes

en distintos escenarios de aplicación de SOA. Gracias a esta arquitectura, cuando dichas situaciones ocurran a partir de la detección de ciertos patrones de eventos predefinidos en el motor CEP, las alertas o alarmas (eventos complejos) serán enviadas en tiempo real a los usuarios y sistemas que estén interesados en recibirlas.

El elemento principal de la arquitectura es un ESB que servirá como nexo de unión entre los productores de eventos, los consumidores de eventos y el motor CEP. Dado que la arquitectura permite la integración con una gran diversidad de productores y consumidores de eventos, podrá utilizarse en múltiples dominios de aplicación.

Entre sus funciones principales se encuentran la recepción de eventos, la transformación de eventos, el filtrado de eventos, la generación dinámica del dominio, el envío de eventos al motor CEP, la adición de patrones de eventos en el motor CEP, la adición de acciones para patrones de eventos en el ESB, la detección de patrones de eventos, la recepción de eventos complejos y la toma de acciones (acciones a llevar a cabo al detectar los eventos complejos). De todas ellas, las funcionalidades más relevantes con respecto a otras arquitecturas existentes son la integración de Mule con Esper, la generación automática del domino y su integración con los editores gráficos de modelado de dominio CEP y de patrones de eventos desarrollados en esta tesis doctoral, así como la inserción automática y en tiempo de ejecución tanto del código EPL de los patrones de eventos en el motor CEP como del código XML de las acciones a ejecutar en el ESB cuando se detecten los patrones; todo ello sin que se requiera ninguna modificación adicional en la implementación de la arquitectura propuesta.

Capítulo 7

Casos de Estudio

«La informática es la ciencia de la abstracción —se crea el modelo correcto para un problema y se inventa las técnicas mecanizadas apropiadas para resolverlo.»
(Alfred Vaino Aho y Jeff Ullman)

En este capítulo se presentan dos casos de estudio en los que se aplica el enfoque propuesto en el Capítulo 3 a dos dominios de aplicación. El primer caso de estudio se basa en la detección de situaciones críticas y relevantes en el ámbito de la domótica, mientras que el segundo caso de estudio está orientado a la detección de dichas situaciones en el ámbito de la seguridad en redes.

7.1. Caso de Estudio sobre Domótica

En los últimos años está creciendo el número de casas inteligentes que disponen de sensores de datos para monitorizar las condiciones del entorno en el que se encuentran. Estos sensores pueden medir con cierta frecuencia la temperatura y humedad externa o interna de un hogar, o el consumo energético o de gas, entre otros. Además, existe una tendencia de compartir esta información recogida por dichos sensores en plataformas IoT de acceso libre.

Este escenario cobra interés a la hora de monitorizar estos datos en tiempo real para detectar situaciones relevantes o críticas que sucedan en hogares repartidos por todo el mundo; tratando de paliar cuanto antes sus consecuencias. Por ejemplo, sería deseable que se avisase a los habitantes de una casa cuando realicen un consumo irresponsable de electricidad puesto que, de esta forma, podrán poner más cuidado en cómo usar sus electrodomésticos.

Para la consecución de estos objetivos, en este caso de estudio se hace uso del enfoque propuesto en el Capítulo 3 y, más específicamente, se aplica la propuesta de integración de CEP en SOA 2.0, presentada en el Capítulo 6, al ámbito de la domótica, en la que se utiliza como productor de eventos Xively [Log14], una de las plataformas IoT bien

documentada y altamente escalable, que gestiona cada día millones de datos provenientes de miles de personas, organizaciones y compañías a nivel mundial. Su finalidad es permitir almacenar, compartir y extraer datos en tiempo real ofrecidos por los distintos objetos y dispositivos distribuidos geográficamente a nivel mundial. Para ello, ofrece una API (*Application Programming Interface*) RESTful —API de servicios web basada en REST— que proporciona en tiempo real los valores observados como flujos de datos en formato XML, CSV (*Comma-Separated Values*) o JSON.

Por otra parte, como consumidores de eventos se utilizan en este caso de estudio tanto servicios de correo electrónico como servicios de redes sociales; concretamente, el servicio de correos de la UCA (*Universidad de Cádiz*) y Twitter, respectivamente. Cabe recordar que la arquitectura SOA 2.0 propuesta en esta tesis está implementada de forma que la adición de otros tipos de consumidores de eventos sea totalmente viable cuando así se necesite.

A lo largo de esta sección se detallan, en primer lugar, la implementación del flujo Mule que permite obtener y gestionar los eventos desde la plataforma Xively y su integración con la arquitectura descrita en el Capítulo 6, seguido de los pasos a llevar a cabo, según el enfoque definido en el Capítulo 3, desde que el experto en el dominio define el dominio sobre domótica hasta que los usuarios finales —los habitantes de los hogares en cuestión— reciben las notificaciones en tiempo real de las situaciones acontecidas en las que estén interesados. Estas situaciones podrán ser algunas de las siguientes: consumo energético irresponsable, incendio, corte eléctrico y olvido del apagado de una televisión.

7.1.1. Recepción y Gestión de Eventos Provenientes de una Plataforma IoT

Como se ha comentado previamente, en este caso de estudio se utilizan algunos de los datos proporcionados por Xively. Concretamente, se ha realizado un estudio de cuáles de los flujos de datos disponibles en dicha plataforma sobre domótica son más idóneos para la monitorización, teniendo en cuenta el número de sensores disponibles por hogar, así como la frecuencia de actualización de los datos. En la Tabla 7.1 se muestran los nombres de los flujos escogidos, el país donde se encuentran los sensores asociados a cada flujo, las URL de estos flujos, así como la frecuencia de actualización de datos de cada uno de ellos.

Tabla 7.1: Flujos de Xively sobre domótica utilizados en este caso de estudio.

Nº	**Nombre del flujo**	**País**	**URL**	**Act.**
F1	*Residential Information*	Holanda	`https://xively.com/feeds/62988`	1 min
F2	*Home Automation*	Bélgica	`https://xively.com/feeds/71257`	1 min
F3	*Current Cost Bridge*	España	`https://xively.com/feeds/89125`	5 min

Debe tenerse en cuenta que en cada uno de estos flujos se depositan los valores tomados de los distintos sensores que se encuentren en el hogar asociado a ese flujo. Por tanto, el nombre y el número de tipos de datos proporcionados por cada flujo es variable e

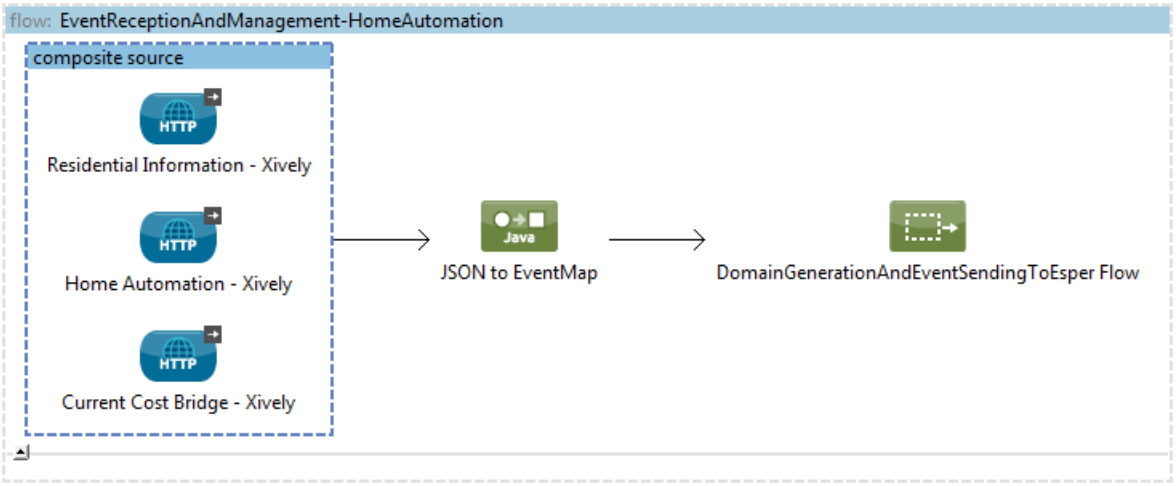

Figura 7.1: Implementación del flujo de recepción y gestión de eventos del caso de estudio sobre domótica.

incluso algunos de ellos podrían presentarse en distintas unidades. Por este motivo, ha sido necesario normalizar los datos de estos tres flujos (F1 - F3), como se explicará a continuación.

Así pues, con el propósito de obtener los datos de dichos flujos y normalizarlos para posteriormente detectar situaciones críticas y/o relevantes sobre domótica, se ilustra en la Figura 7.1 el único flujo, denominado *EventReceptionAndManagement-HomeAutomation*, que ha sido necesario implementar adicionalmente a la propuesta ya definida para la integración de SOA 2.0 y CEP (véase las Figuras 6.2 y 6.3). Esto pone de manifiesto que dicha solución puede ser extendida a distintos dominios de aplicación, simplemente incorporando un nuevo flujo específico para la recepción y la gestión de la información para el dominio en cuestión.

Como puede observarse, el flujo *EventReceptionAndManagement-HomeAutomation* dispone de tres *endpoints* HTTP de entrada, un transformador de formato JSON a Map y, finalmente, un componente de referencia al flujo *DomainDynamicGenerationAndEventSendingToEsper*.

En primer lugar, se ha utilizado un *composite source*, un ámbito Mule donde dos o más fuentes de mensajes pueden posicionarse con la intención de aceptar mensajes provenientes de distintas fuentes externas. En concreto, en este flujo se han utilizado tres *endpoints* HTTP de entrada en los que se han configurado las URLs para los flujos Xively descritos en la Tabla 7.1, el formato en el que se desea obtener la información (JSON en este caso), las credenciales para tener acceso a los datos del servidor y los conectores *polling* indicando cada cuánto tiempo se va obtener la información. En particular, un *polling* que se encarga de obtener datos con una frecuencia de actualización de 1 minuto para los flujos F1 y F2, y otro *polling* con frecuencia de 5 minutos para el flujo F3.

Para convertir los datos obtenidos en formato JSON a eventos de tipo *map* de Java

así como para normalizarlos, se ha creado un transformador en Java denominado *JSON to EventMap*. En cuanto a la normalización, en la Tabla 7.2 se especifica el nombre que se le ha asignado a cada dato junto con su tipo, una descripción y si se encuentra disponible el dato en ese flujo (marcado con una *x* en caso afirmativo).

Tabla 7.2: Formato de los datos de flujos normalizados.

Dato	**Tipo**	**Descripción**	**F1**	**F2**	**F3**
home	*String*	Nombre del flujo.	x	x	x
sensor	*String*	URL del flujo.	x	x	x
location	*Location*	Localización del hogar.	x	x	x
location.name	*Float*	Nombre de la ciudad, país.	x	x	x
location.latitude	*Float*	Latitud de la localización.	x	x	x
location.longitude	*Float*	Longitud de la localización.	x	x	x
timestamp	*String*	Fecha y hora de registro del dato.	x	x	x
energyConsumption	*Float*	Consumo energético en el hogar (W).	x	x	x
temperature	*Float*	Temperatura del hogar (^{o}C).	x	x	x
humidity	*Float*	Humedad del hogar (%).	x	x	
tvConsumption	*Float*	Consumo de la televisión (W).			x

Con la finalidad de simplificar la implementación del transformador, se ha utiliza Jackson [Jac14], una biblioteca Java multipropósito para procesar datos en formato JSON. Este transformador obtiene el cuerpo (*payload*) del mensaje de tipo *MuleMessage* recibido en este transformador y, a partir de la información contenida en este *payload*, se creará y devolverá un nuevo *map* de Java, denominado *eventMap* de tipo *Map<String, Object>*. La clave de tipo *String* almacenará el nombre del evento (*HomeEvent*) mientras que el valor de tipo *Object* será, a su vez, otro *map* de tipo *Map<String, Object>*, denominado *eventPayload*, que almacenará todas las propiedades de dicho evento con sus respectivos valores como, por ejemplo, *sensor* y *location*. La propiedad *location*, al ser una propiedad compuesta por las propiedades *name*, *latitude* y *longitude*, será, a su vez, otro *map* anidado.

Finalmente, el componente de referencia de flujo enviará cada *eventMap* recibido con los datos en el formato apropiado al flujo *DomainDynamicGenerationAndEventSending-ToEsper*, el cual se encargará de generar el dominio CEP automáticamente así como enviar los eventos *HomeEvent* al motor CEP para su procesamiento.

Tras la incorporación de este flujo, la arquitectura ya podrá analizar y procesar información proveniente de la plataforma Xively. Sin embargo, si lo que se pretende es, además, detectar situaciones críticas y/o relevantes para el ámbito de la domótica y notificarlas a los sistemas y usuarios interesados, entonces es necesario incorporar al motor CEP los patrones de eventos que describirán las condiciones que deben cumplirse para llevar a cabo dicha detección. A continuación, se detallan los pasos a seguir para lograrlo.

7.1.2. Definición del Modelo de Dominio CEP

Un experto en el dominio domótico será el responsable de definir gráficamente el dominio CEP conforme al metamodelo descrito en la Sección 4.2.1. Este dominio estará compuesto por los tipos de eventos y propiedades que describan el dominio sobre domótica; en este caso, el tipo de evento *HomeEvent* con sus propiedades (véase la Tabla 7.2).

Esta tarea podrá ser realizada por el experto en el dominio bien utilizando el editor gráfico de dominios CEP (véase el Capítulo 4) o bien directamente mediante el editor gráfico de patrones de eventos (véase el Capítulo 5), ya que este editor, además de ofrecer sus funcionalidades propias de gestión de patrones, también permite la definición de un modelo de dominio CEP, siempre y cuando el editor todavía no haya sido reconfigurado para un dominio en particular. Se ha optado por utilizar el editor de propósito específico para la definición de dominios.

Es importante recordar que el dominio CEP puede crearse de dos formas: manualmente —opción del menú *CEP Domain > New...*— lo que implica que el experto debe definir el tipo de evento *HomeEvent* y sus propiedades haciendo uso de las herramientas *Event* y *EventProperty* disponibles en la paleta del editor; dinámicamente —opción del menú *CEP Domain > Auto-detect Domain*— que generará automáticamente el dominio gráfico a partir del tipo de eventos inferido desde los eventos que viajen por el flujo *DomainDynamicGenerationAndEventSendingToEsper* del ESB.

En este caso de estudio, se ha generado automáticamente el dominio realizándose seguidamente algunas modificaciones sobre el mismo. En particular, se ha asignado el nombre *home-automation* al dominio formado únicamente por el tipo de eventos inferido —*HomeEvent*—, así como una descripción textual. Además, se ha asociado este tipo de eventos con un icono gráfico para mejorar la usabilidad, de modo que cualquier usuario pueda reconocerlo de una forma intuitiva. Asimismo, se han asignado iconos a la mayoría de las propiedades del evento, como puede comprobarse en la Figura 7.2. Si no se hubiese asignado un icono para el tipo de eventos aparecería un icono en verde con la letra E, mientras que se usa, por defecto, un icono en verde con la letra P para las propiedades; como ejemplo, véanse las propiedades anidadas *name*, *latitude* y *longitude* de la propiedad *location* a las que no se les han asignado ningún icono. Téngase en cuenta que visualmente podría ocultarse estas propiedades haciendo clic dentro de la propiedad *location*.

7.1.3. Validación y Almacenamiento del Modelo de Dominio CEP

Una vez modelado el dominio de domótica, se ha validado automáticamente a partir de las restricciones definidas en la Sección 4.2.1, y se ha guardado en el sistema tras comprobarse que el modelo es correcto y que no incumple ninguna de las restricciones. Para ejecutar estas funcionalidades se ha seleccionado la opción del menú *CEP Domain > Save and Validate* del editor gráfico de dominios.

A continuación, se ha procedido a la exportación de este dominio CEP. Para ello, se ha utilizado la opción del menú *File > Export CEP Domain*, obteniéndose un archivo

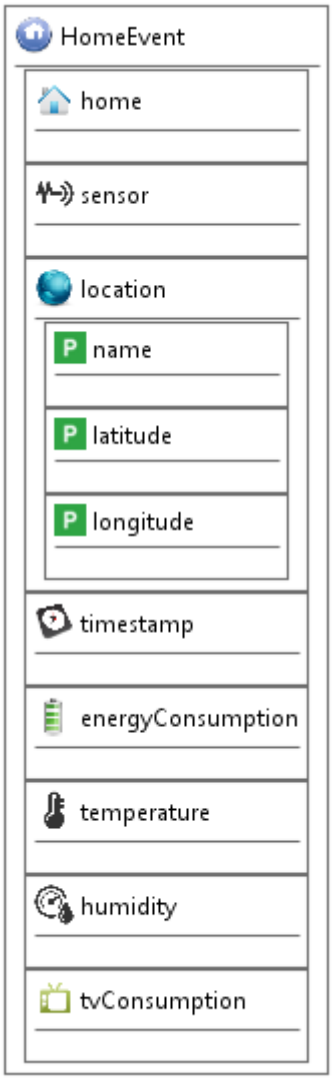

Figura 7.2: Dominio CEP de domótica.

ZIP, denominado *home-automation_domain.zip*, conteniendo tanto el modelo de dominio y el fichero de diagrama como las imágenes de los iconos a las que haga referencia dicho modelo.

A partir de este momento, los usuarios que necesiten definir patrones de eventos para el ámbito de la domótica, podrán importar este dominio *home-automation* en el editor gráfico reconfigurable para el modelado y la generación de código de patrones de eventos.

7.1.4. Definición de Modelos de Patrón de Eventos

Para que la definición gráfica de patrones de eventos sobre domótica pueda llevarse a cabo, es necesario que previamente se haya reconfigurado el editor gráfico de patrones con el dominio *home-automation* recién creado. Al importarse, se añadirá automáticamente el tipo de eventos *HomeEvent* como una herramienta más en la paleta del editor, posicionándose en el grupo de herramientas denominado *Simple Events* (véase la Figura 7.3). Así pues, cuando el usuario necesite describir alguna condición de patrón donde *HomeEvent* deba estar presente, hará uso de dicha herramienta para añadir este tipo de eventos en el canvas del editor.

Gracias a esta personalización dinámica de la paleta del editor, se aumenta considerablemente la usabilidad y experiencia del usuario final al no tener que preocuparse de cómo representar el tipo de eventos y sus propiedades; simplemente deberá usarlo tal cual sea visualizado en el canvas. Por otro lado, esto proporciona otra ventaja fundamental: el usuario final no puede realizar ninguna modificación sobre el tipo de eventos dado, ya que esto conllevaría a la creación de patrones de eventos inconsistentes, al usarse definiciones distintas para un mismo tipo de eventos.

A continuación, se detallan los patrones de eventos definidos gráficamente en este caso de estudio para la detección de situaciones críticas y/o relevantes en el ámbito de la domótica.

Patrón de Consumo Energético Irresponsable

A continuación, se describe el patrón de consumo energético irresponsable que se pretende modelar con el editor gráfico de patrones de eventos:

- **Nombre**: *IrresponsibleEnergyConsumption*.
- **Descripción**: Este patrón permite detectar cuándo se está realizando un alto consumo energético en un hogar y en un intervalo de tiempo determinado.
- **Condiciones del patrón**:
    - Cada evento de tipo *HomeEvent* cuyo consumo energético (propiedad *energyComsuption*) sea superior a 1500 vatios. Es importante señalar que el valor de 1500 vatios podría ser sustituido por otro, si así lo estima oportuno el experto en domótica.

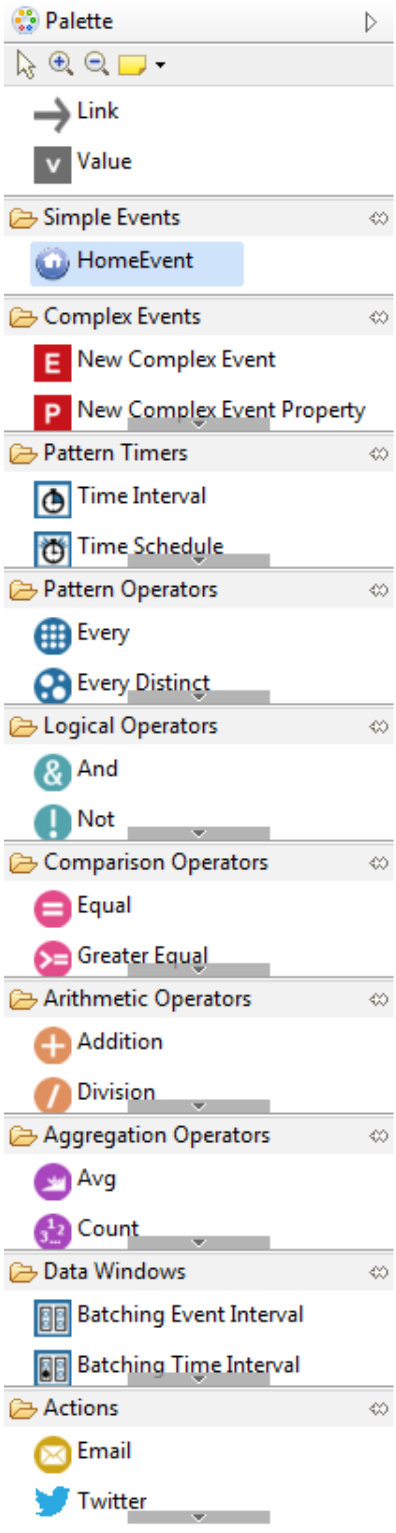

Figura 7.3: Paleta del editor reconfigurable personalizada con el dominio de domótica.

  - Y, además, se lleve a cabo en un intervalo de tiempo de 10 minutos.

- **Nuevo evento complejo**: *IrresponsibleEnergyConsumption*. Se creará un nuevo evento complejo con las siguientes propiedades:

  - *home*: el hogar en el que se ha detectado el consumo energético irresponsable.
  - *locationName*: la ubicación del hogar.
  - *timestamp*: la fecha y la hora en la que se ha producido el evento *HomeEvent*.
  - *energyConsumption*: el consumo energético detectado en ese momento. Este consumo será superior a 1500 vatios, puesto que es la condición que se ha establecido en el patrón.

- **Acciones:**

  - *Email*: cada nuevo evento complejo de tipo *IrresponsibleEnergyConsumption* creado, tras el cumplimiento de las condiciones del patrón, será enviado a la dirección o direcciones de correo electrónico especificadas en el componente *Email* (atributo *To*). Para ello, deberá indicarse también información sobre la dirección del emisor del mensaje, el *host* y el puerto, el nombre de usuario y contraseña, así como el asunto del correo, por ejemplo, *Alert: Irresponsible Energy Consumption*.
  - *Twitter*: cada nuevo evento complejo también será enviado a una cuenta de Twitter en la que se registrarán todas las situaciones acontecidas sobre el dominio domótico.

La Figura 7.4 presenta el modelo de este patrón diseñado con el editor de patrones. Cabe destacar que el operador azul con nueve puntos en blanco se trata del operador de patrón *Every*, encargado de seleccionar *cada uno* de los eventos de tipo *HomeEvent* que cumpla la condición especificada en cuanto al consumo energético. Para más información sobre la descripción de los operandos y operadores de patrones, consúltese la Sección 5.2.1. Como puede observarse, se ha asignado una imagen al icono que representará al nuevo tipo de evento complejo *IrresponsibleEnergyConsumption*.

Patrón de Incendio

Seguidamente, se describe el patrón de incendio que se pretende modelar con el editor gráfico de patrones de eventos:

- **Nombre**: *Fire*.
- **Descripción**: Este patrón permite detectar un posible caso de incendio en un hogar.
- **Condiciones del patrón**:

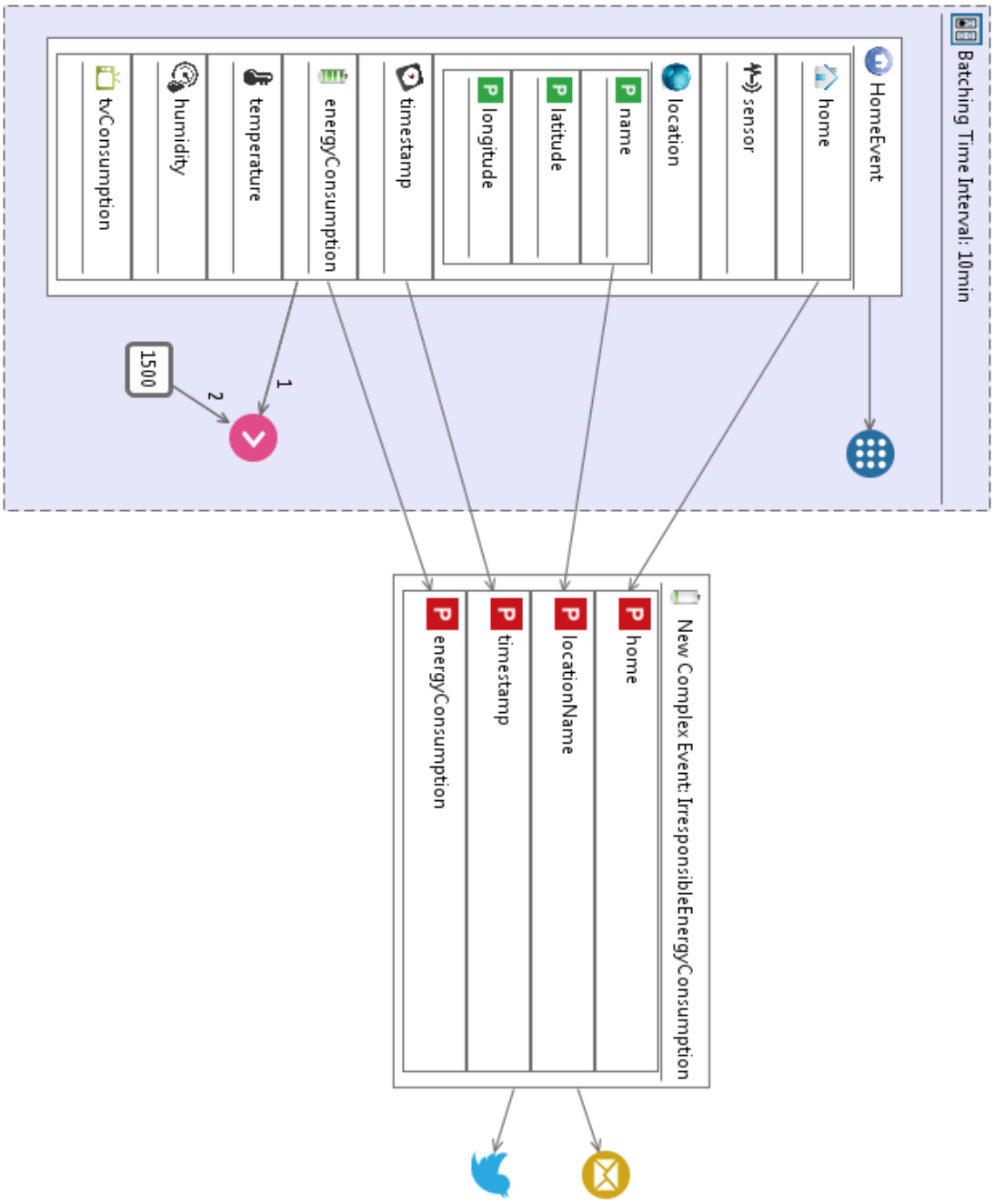

Figura 7.4: Modelo del patrón de consumo energético irresponsable.

  - Una vez analizada la temperatura de un hogar, si en el minuto siguiente se observa que la temperatura ha aumentado en más de 20^{o}C en ese mismo hogar, entonces se puede deducir que se ha producido un incendio. Lógicamente, es improbable que en tan solo un minuto la temperatura pueda oscilar en más de 20^{o}C ni siquiera utilizando una estufa para calentar el ambiente; salvo que se produzca algún incendio en la casa.

- **Nuevo evento complejo**: *Fire*. Se creará un nuevo evento complejo con las siguientes propiedades:

  - *home*: el hogar en el que se ha detectado el caso de incendio.
  - *locationName*: la ubicación del hogar.
  - *timestamp*: la fecha y la hora en la que se ha producido el evento *HomeEvent*.
  - *initialTemperature*: la temperatura correspondiente al primero de los dos eventos implicados en el cumplimiento de las condiciones descritas.
  - *finalTemperature*: la temperatura correspondiente al segundo de los dos eventos implicados en el cumplimiento de las condiciones descritas. El valor de esta temperatura superará los 20^{o}C con respecto a la temperatura referida como temperatura inicial.

- **Acciones:**

  - *Twitter*: cada nuevo evento complejo será enviado a una cuenta de Twitter en la que se registrarán todas las situaciones acontecidas sobre el dominio domótico.

La Figura 7.5 ilustra el modelo de este patrón diseñado con el editor de patrones. Cabe destacar que el operador azul con una flecha en blanco se trata del operador de patrón *Followed By*, que define una relación de orden donde se establece que la primera expresión de patrón debe ser seguida de otra expresión, mientras que el operador cuadrado y azul es el temporizador de patrón *Time Interval*. El operador *Every* hará posible que se cree un nuevo evento complejo cada vez que se detecten dichas condiciones. Además, se ha asignado una imagen al icono que representará al nuevo tipo de evento complejo *Fire*.

Patrón de Corte Eléctrico

A continuación, se describe el patrón de corte eléctrico que se pretende modelar con el editor gráfico de patrones de eventos:

- **Nombre**: *PowerFailure*.

- **Descripción**: Este patrón avisará cuando se produzca un corte del suministro eléctrico en un hogar.

- **Condiciones del patrón**:

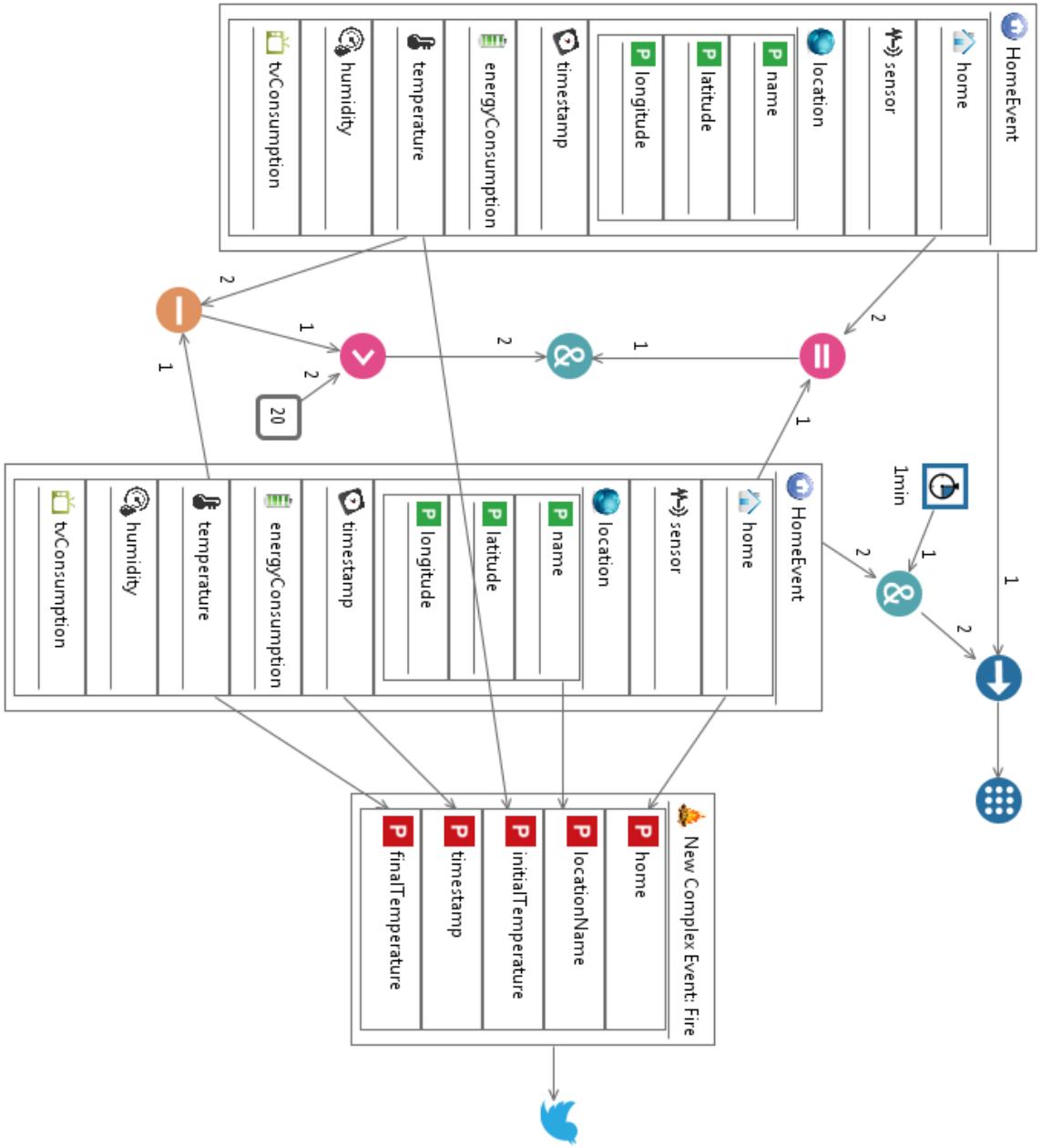

Figura 7.5: Modelo del patrón de incendio.

  - En un momento determinado el consumo energético es mayor que 0 vatios y, seguidamente, el consumo energético en ese mismo hogar es igual a 0. Téngase en cuenta que se considera que el consumo energético en un hogar nunca será 0 vatios, ya que siempre existirá algún electrodoméstico enchufado a la toma de corriente como, por ejemplo, un frigorífico.

- **Nuevo evento complejo**: *PowerFailure*. Se creará un nuevo evento complejo con las siguientes propiedades:

  - *home*: el hogar en el que se ha detectado el corte del suministro eléctrico.
  - *locationName*: la ubicación del hogar.
  - *timestamp*: la fecha y la hora en la que se ha producido el evento *HomeEvent*.
  - *initialEnergyConsumption*: el consumo energético correspondiente al primero de los dos eventos implicados en el cumplimiento de las condiciones descritas.
  - *finalEnergyConsumption*: el consumo energético correspondiente al segundo de los dos eventos implicados en el cumplimiento de las condiciones descritas.

- **Acciones:**

  - *Email*: cada nuevo evento complejo de tipo *PowerFailure* creado, tras el cumplimiento de las condiciones del patrón, será enviado a la dirección o direcciones de correo electrónico especificadas en el componente *Email*.

La Figura 7.6 muestra el modelo de este patrón diseñado con el editor de patrones. Como puede comprobarse, se ha asignado una imagen al icono que representará al nuevo tipo de evento complejo *PowerFailure*.

Patrón de Olvido del Apagado de una Televisión

Seguidamente, se describe el patrón de olvido del apagado de una televisión que se pretende modelar con el editor gráfico de patrones de eventos:

- **Nombre**: *IrresponsibleTelevisionUse*.

- **Descripción**: Este patrón trata de detectar la situación en la que los habitantes de un hogar han olvidado apagar la televisión antes de salir de casa.

- **Condiciones del patrón**:

  - Si la televisión de un hogar permanece encendida ininterrumpidamente durante 6 horas. En esta ocasión, se presupone que los telespectadores no pasan más de 6 horas seguidas delante del televisor. En caso contrario, habría que aumentar el número de horas impuestas en este patrón.

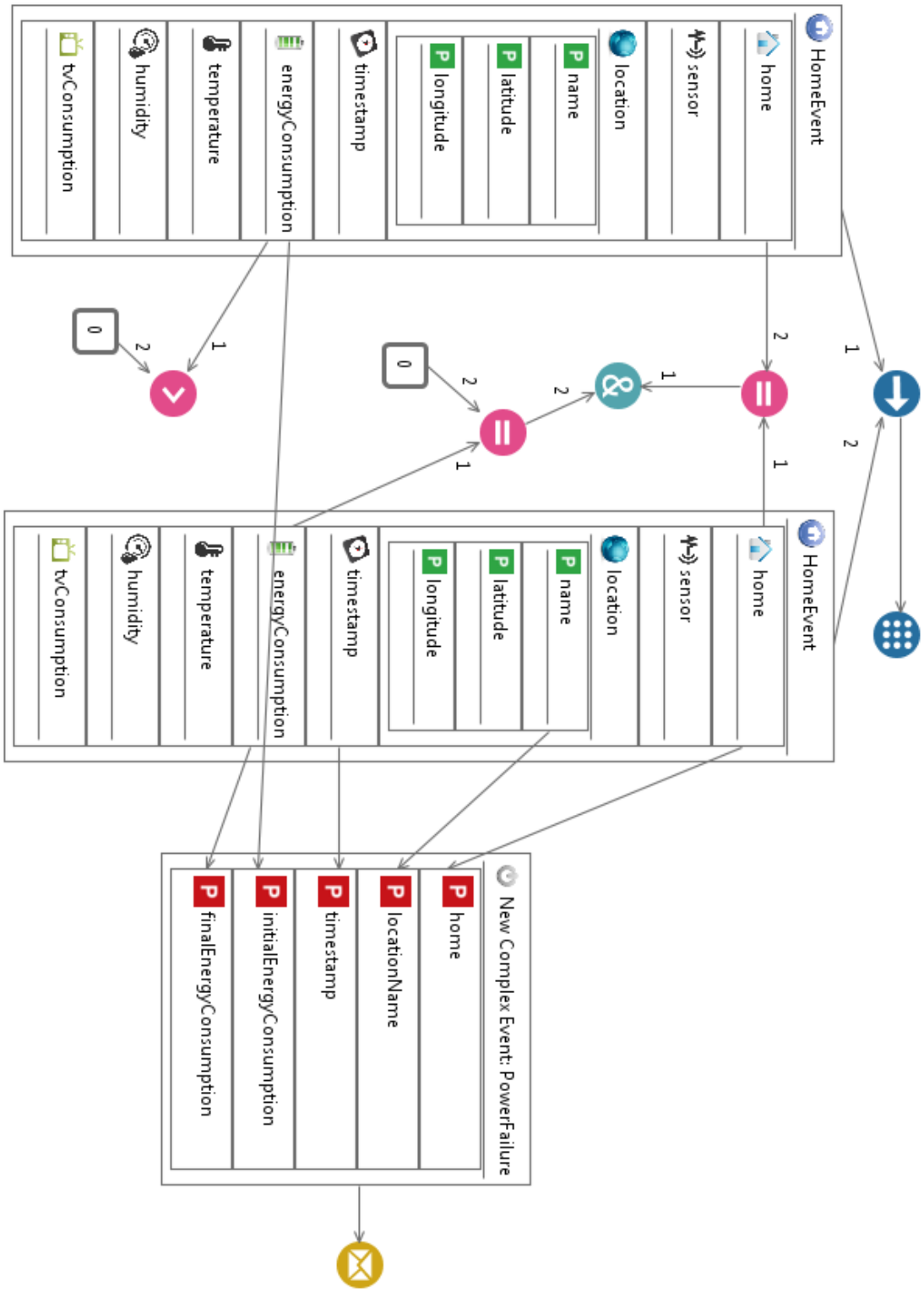

Figura 7.6: Modelo del patrón de corte eléctrico.

- **Nuevo evento complejo**: *IrresponsibleTelevisionUse*. Se creará un nuevo evento complejo con las siguientes propiedades:
  - *home*: el hogar en el que se ha detectado el olvido del apagado de una televisión.
  - *locationName*: la ubicación del hogar.
  - *timestamp*: la fecha y la hora en la que se ha producido el evento *HomeEvent*.
- **Acciones:**
  - *Email*: cada nuevo evento complejo de tipo *IrresponsibleTelevisionUse* creado será enviado a la dirección o direcciones de correo electrónico especificadas en el componente *Email*.

La Figura 7.7 ilustra el modelo de este patrón diseñado con el editor de patrones. Al igual que en los patrones anteriores, se ha asignado una imagen al icono que representará al nuevo tipo de evento complejo *IrresponsibleTelevisionUse*.

7.1.5. Validación y Almacenamiento de Modelos de Patrón de Eventos

Conforme se ha ido modelando cada patrón de domótica detallado anteriormente, se ha validado automáticamente a partir de las restricciones definidas en la Sección 5.2.1, y se ha guardado en el sistema tras comprobarse que el modelo es correcto y que no incumple ninguna de las restricciones. Para ejecutar estas funcionalidades se ha seleccionado la opción del menú *Event Patterns > Save and Validate* del editor gráfico de patrones.

Tras el almacenamiento de cada modelo de patrón en el sistema, se ha reconfigurado automáticamente la paleta del editor, añadiéndose el nuevo tipo de evento complejo definido en cada modelo como una herramienta más dentro del grupo de herramientas denominado *Complex Events* (véase la Figura 7.8). Por lo tanto, cuando el usuario necesite describir alguna condición de patrón en la cual deba estar presente alguno de los tipos de eventos complejos diseñados previamente, hará uso de las herramientas de eventos complejos de la paleta del editor para añadir estos tipos de eventos en el canvas del editor. Lógicamente, no estará disponible la herramienta del tipo de evento complejo en la paleta del propio modelo que en ese momento se esté diseñando, ya que no tendría sentido utilizar un tipo de evento complejo en el mismo modelo donde se defina.

Posteriormente, se ha procedido a la exportación de dichos patrones de eventos. Para ello, se ha utilizado la opción del menú *File > Export Event Patterns*, obteniéndose un archivo ZIP, denominado *home-automation_patterns.zip*, que contiene tanto los modelos de los patrones de eventos (modelos EMF) y sus ficheros de diagrama (diagramas) junto con las imágenes a las que hagan referencia dichos modelos de patrón, como el modelo de dominio (modelo EMF) para el que se ha definido estos patrones, su fichero de diagrama (diagrama) y las imágenes a las que haga referencia dicho modelo de dominio.

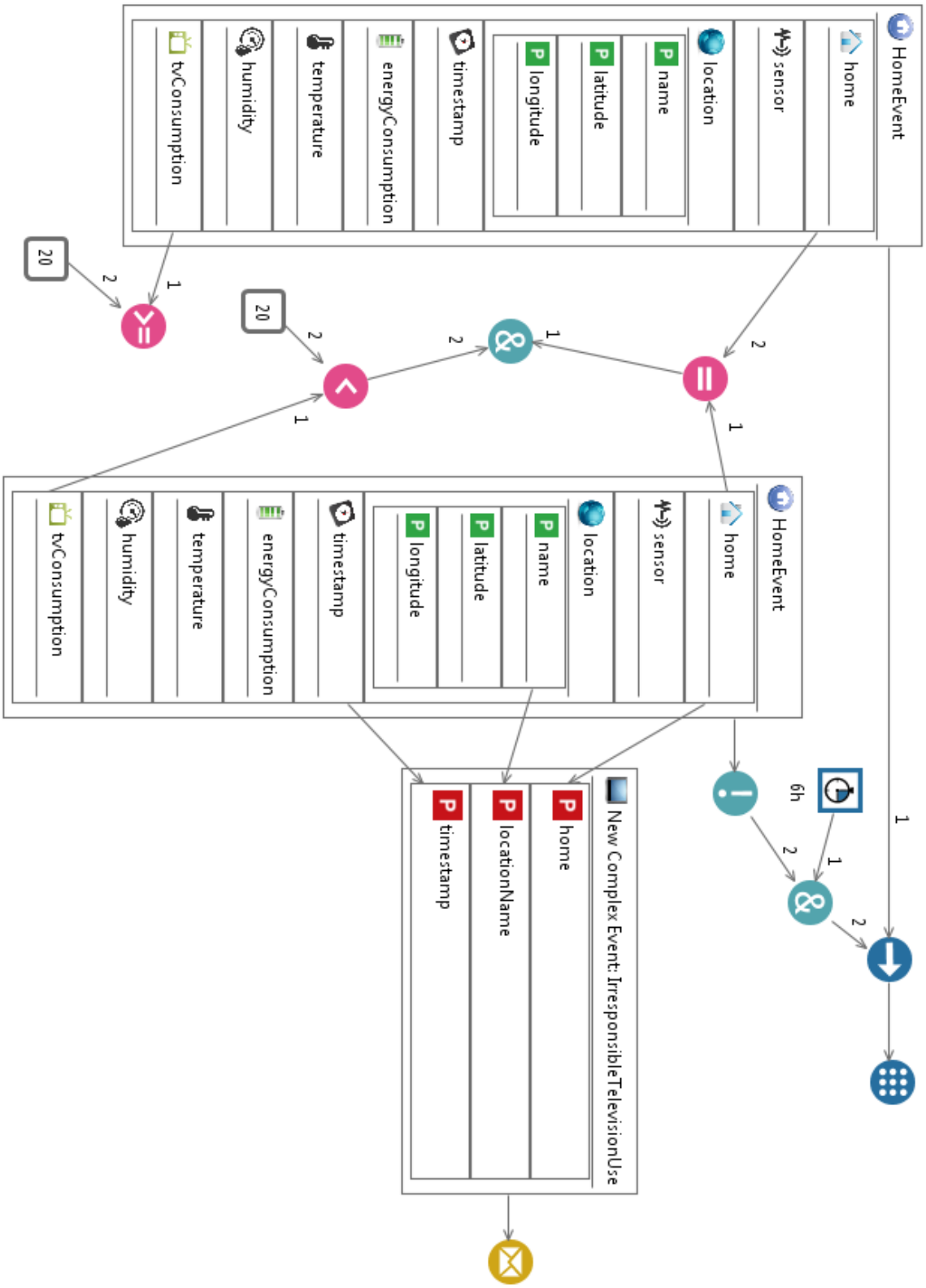

Figura 7.7: Modelo del patrón de olvido del apagado de una televisión.

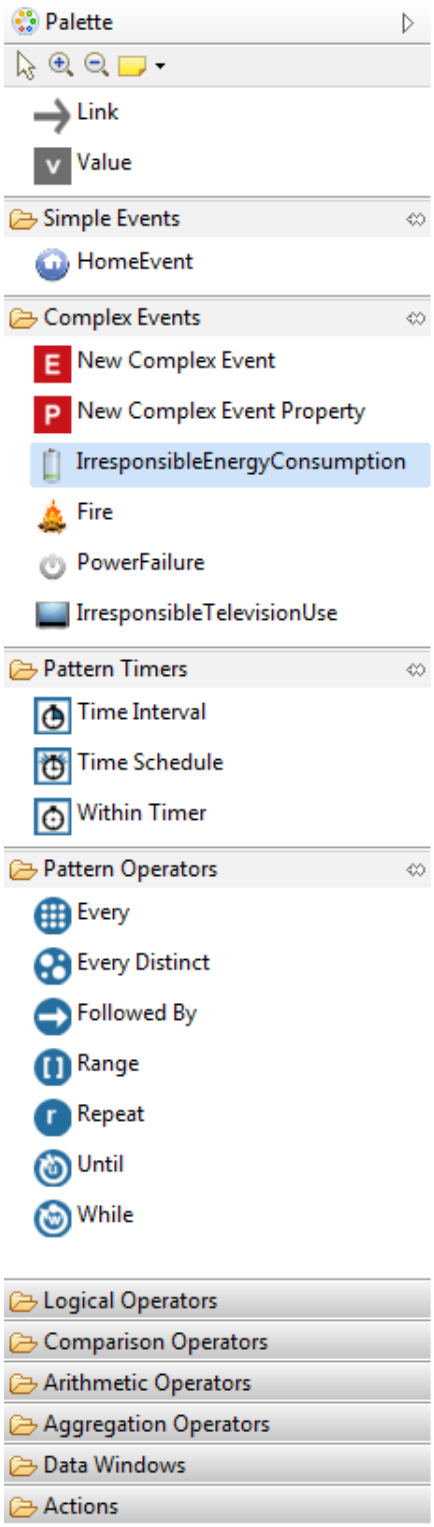

Figura 7.8: Paleta del editor reconfigurable personalizada con los eventos complejos de domótica.

7.1.6. Transformación de Modelos de Patrón de Eventos a Código

Tras el diseño, la validación y el almacenamiento de los patrones de eventos podrá generarse de forma automática tanto el código EPL de los patrones que deben ser añadidos en tiempo de ejecución en el motor Esper, como el código XML correspondiente a las acciones que deben llevarse a cabo en el ESB Mule cuando se detecten dichos patrones. Para lograrlo, deberá utilizarse la opción del menú del editor *Event Patterns > Generate and Deploy Pattern Code*.

Al utilizar esta opción, se pedirá al usuario que seleccione el directorio donde desea que el editor cree los ficheros con extensión *.epl* con el código de los patrones, y el directorio donde desea que se registren los ficheros *.action* con las acciones de los patrones.

A continuación, se presentan los ficheros *.epl* y *.action* generados automáticamente para cada uno de los patrones de domótica definidos y modelados en la Sección 7.1.4.

Patrón de Consumo Energético Irresponsable

Este patrón permite detectar cuándo se está realizando un alto consumo energético en un hogar en un intervalo de tiempo determinado. El Listado 7.1 muestra el código EPL generado para este patrón.

Listado 7.1: Fichero *IrresponsibleEnergyConsumption.epl* generado por el editor de patrones.

```
@Name('IrresponsibleEnergyConsumption')
insert into IrresponsibleEnergyConsumption
select a1.home as home,
   a1.location.name as locationName,
   a1.timestamp as timestamp,
   a1.energyConsumption as energyConsumption
from pattern [every a1 = HomeEvent(a1.energyConsumption > 1500)].
    win:time_batch(10 minutes)
```

En primer lugar, se define el patrón de eventos complejos mediante la cláusula `from pattern`. Se toma de entre todos los eventos de tipo `HomeEvent` aquellos que cumplan la condición en la que el consumo energético en ese instante sea mayor a 1500 W y, a continuación, se le aplica el operador `every` para seleccionar cada uno de estos eventos que representan un alto consumo energético. Para poder seleccionar a posteriori las propiedades de este evento, es necesario asignarle un alias (en este caso denominado `a1`).

Como puede observarse, a este patrón se le ha aplicado además una ventana temporal `win:time_batch(10 min)`, esto quiere decir que aunque se detecte varias veces el patrón de consumo energético irresponsable durante el intervalo de tiempo de 10 minutos, solo se notificará que el patrón ha sido detectado al finalizar los 10 minutos establecidos.

Finalmente, para cada uno de los eventos que cumple la condición impuesta en el patrón, se seleccionan las siguientes propiedades: el hogar donde se ha detectado la situación y su localización, el tiempo de registro, y el consumo energéti-

co detectado en ese momento. Con estas propiedades se crea un nuevo evento complejo de tipo `IrresponsibleEnergyConsumption(home, locationName, timestamp, energyConsumption)` y se inserta en un nuevo flujo de eventos de Esper con la misma denominación que el evento complejo.

Por otro lado, el Listado 7.2 presenta el código XML correspondiente a las acciones definidas gráficamente para el patrón de consumo energético irresponsable. En la primera línea del fichero se indica la declaración XML, seguida de la declaración de los espacios de nombre y la localización del esquema de Mule.

El código de las acciones será añadido en el ESB Mule en tiempo de ejecución como un nuevo flujo (<flow>) dinámico denominado igual que su patrón correspondiente: `IrresponsibleEnergyConsumption`. Este flujo está compuesto por el encaminador de mensajes `All` proporcionado por Mule, que se encargará de difundir todos los eventos complejos que se reciban en el flujo Mule denominado *ComplexEventReceptionAndDecisionMaking* (véase la Figura 6.3), tanto por correo electrónico a la dirección indicada como a la cuenta Twitter creada para tal fin. Nótese que previo al envío del correo electrónico, se ha establecido el formato del cuerpo del mensaje mediante <set-payload>, y que el componente Twitter referencia a un componente global denominado `Twitter`, donde se encuentran registradas las claves de usuario y consumidor obtenidas al registrar en `https://dev.twitter.com` la aplicación Mule implementada en esta tesis doctoral.

Listado 7.2: Fichero *IrresponsibleEnergyConsumption.action* generado por el editor de patrones.

```
<?xml version="1.0" encoding="UTF-8"?>

<mule xmlns:twitter="http://www.mulesoft.org/schema/mule/twitter"
    xmlns:smtps="http://www.mulesoft.org/schema/mule/smtps"
    xmlns:smtp="http://www.mulesoft.org/schema/mule/smtp"
    xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
    xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
    xmlns:spring="http://www.springframework.org/schema/beans" version="EE
        -3.4.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://
        www.springframework.org/schema/beans/spring-beans-current.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/
            schema/mule/core/current/mule.xsd
        http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/
            schema/mule/vm/current/mule-vm.xsd
        http://www.mulesoft.org/schema/mule/smtp http://www.mulesoft.org/
            schema/mule/smtp/current/mule-smtp.xsd
        http://www.mulesoft.org/schema/mule/smtps http://www.mulesoft.org/
            schema/mule/smtps/current/mule-smtps.xsd
        http://www.mulesoft.org/schema/mule/twitter http://www.mulesoft.org/
            schema/mule/twitter/2.4/mule-twitter.xsd">
```

```
    <flow name="IrresponsibleEnergyConsumption">
        <all doc:name="All">
            <processor-chain>
                <set-payload value="Detected Alert '#[message.
                    inboundProperties['eventPatternName']]': #[payload]"
                    doc:name="Set Email Body"/>
                <smtps:outbound-endpoint host="smtp.uca.es" port="465" user=
                    "****" password="****" to="juan.boubeta@uca.es" from="
                    juan.boubeta@uca.es" cc="" subject="Alert: Irresponsible
                    Energy Consumption" responseTimeout="3000" doc:name="SMTP
                    "/>
            </processor-chain>

            <twitter:update-status config-ref="Twitter" status="Detected
                Alert '#[message.inboundProperties['eventPatternName']]' (#[
                message.id])" doc:name="Twitter"/>
        </all>
    </flow>

</mule>
```

Patrón de Incendio

Este patrón permite detectar un posible caso de incendio en un hogar. El Listado 7.3 muestra el código EPL generado para este patrón, donde se comprueba si una vez analizada la temperatura de un hogar, en el minuto siguiente se observa que la temperatura ha aumentado en más de 20^{o}C. En caso afirmativo se creará un evento complejo de tipo `Fire(home, locationName, initialTemperature, timestamp, finalTemperature)`.

Listado 7.3: Fichero *Fire.epl* generado por el editor de patrones.

```
@Name('Fire')
insert into Fire
select a2.home as home,
   a2.location.name as locationName,
   a1.temperature as initialTemperature,
   a2.timestamp as timestamp,
   a2.temperature as finalTemperature
from pattern [every (a1 = HomeEvent -> (timer:interval(1 minutes) and a2 =
    HomeEvent((a2.home = a1.home and (a2.temperature - a1.temperature) > 20))
    ))]
```

En cuanto al código de las acciones generado por el editor, el Listado 7.4 presenta el código XML con la acción de envío a la cuenta Twitter cuando se detecte un evento complejo de tipo `Fire`.

Listado 7.4: Fichero *Fire.action* generado por el editor de patrones.

```
<?xml version="1.0" encoding="UTF-8"?>

<mule xmlns:twitter="http://www.mulesoft.org/schema/mule/twitter"
```

```
    xmlns:smtps="http://www.mulesoft.org/schema/mule/smtps"
    xmlns:smtp="http://www.mulesoft.org/schema/mule/smtp"
    xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
    xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
    xmlns:spring="http://www.springframework.org/schema/beans" version="EE
        -3.4.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://
        www.springframework.org/schema/beans/spring-beans-current.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/
            schema/mule/core/current/mule.xsd
        http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/
            schema/mule/vm/current/mule-vm.xsd
        http://www.mulesoft.org/schema/mule/smtp http://www.mulesoft.org/
            schema/mule/smtp/current/mule-smtp.xsd
        http://www.mulesoft.org/schema/mule/smtps http://www.mulesoft.org/
            schema/mule/smtps/current/mule-smtps.xsd
        http://www.mulesoft.org/schema/mule/twitter http://www.mulesoft.org/
            schema/mule/twitter/2.4/mule-twitter.xsd">

    <flow name="Fire">

        <twitter:update-status config-ref="Twitter" status="Detected Alert
            '#[message.inboundProperties['eventPatternName']]' (#[message.id
            ])" doc:name="Twitter"/>
    </flow>

</mule>
```

Patrón de Corte Eléctrico

Este patrón avisará cuando se produzca un corte del suministro eléctrico en un hogar. El Listado 7.5 muestra el código EPL generado para este patrón, donde se comprueba si en un momento determinado el consumo energético es mayor que 0 vatios y, seguidamente, el consumo energético en ese mismo hogar es igual a 0. En caso afirmativo se creará un evento complejo de tipo `PowerFailure(home, locationName, timestamp, initialEnergyConsumption, finalEnergyConsumption)`.

Listado 7.5: Fichero *PowerFailure.epl* generado por el editor de patrones.

```
@Name('PowerFailure')
insert into PowerFailure
select a2.home as home,
   a2.location.name as locationName,
   a2.timestamp as timestamp,
   a1.energyConsumption as initialEnergyConsumption,
   a2.energyConsumption as finalEnergyConsumption
from pattern [every (a1 = HomeEvent(a1.energyConsumption > 0) -> a2 =
    HomeEvent((a2.home = a1.home and a2.energyConsumption = 0)))]
```

Por otro lado, el Listado 7.6 presenta el código XML con la acción de envío por correo electrónico cuando se detecte un evento complejo de tipo `PowerFailure`.

Listado 7.6: Fichero *PowerFailure.action* generado por el editor de patrones.

```
<?xml version="1.0" encoding="UTF-8"?>

<mule xmlns:twitter="http://www.mulesoft.org/schema/mule/twitter"
    xmlns:smtps="http://www.mulesoft.org/schema/mule/smtps"
    xmlns:smtp="http://www.mulesoft.org/schema/mule/smtp"
    xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
    xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
    xmlns:spring="http://www.springframework.org/schema/beans" version="EE
        -3.4.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://
        www.springframework.org/schema/beans/spring-beans-current.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/
            schema/mule/core/current/mule.xsd
        http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/
            schema/mule/vm/current/mule-vm.xsd
        http://www.mulesoft.org/schema/mule/smtp http://www.mulesoft.org/
            schema/mule/smtp/current/mule-smtp.xsd
        http://www.mulesoft.org/schema/mule/smtps http://www.mulesoft.org/
            schema/mule/smtps/current/mule-smtps.xsd
        http://www.mulesoft.org/schema/mule/twitter http://www.mulesoft.org/
            schema/mule/twitter/2.4/mule-twitter.xsd">

    <flow name="PowerFailure">
        <processor-chain>
            <set-payload value="Detected Alert '#[message.inboundProperties
                ['eventPatternName']]': #[payload]" doc:name="Set Email Body"
                />
            <smtps:outbound-endpoint host="smtp.uca.es" port="465" user="
                ****" password="****" to="juan.boubeta@uca.es" from="juan.
                boubeta@uca.es" cc="" subject="Alert: PowerFailure"
                responseTimeout="3000" doc:name="SMTP"/>
        </processor-chain>
    </flow>

</mule>
```

Patrón de Olvido del Apagado de una Televisión

Este patrón trata de detectar la situación en la que los habitantes de un hogar han olvidado apagar la televisión antes de salir de casa. El Listado 7.7 muestra el código EPL generado para este patrón, donde se comprueba si la televisión permanece encendida ininterrumpidamente durante 6 horas. En caso afirmativo se creará un evento complejo de tipo `IrresponsibleTelevisionUse(home, locationName, timestamp)`.

Listado 7.7: Fichero *IrresponsibleTelevisionUse.epl* generado por el editor de patrones.

```
@Name('IrresponsibleTelevisionUse')
insert into IrresponsibleTelevisionUse
select a2.home as home,
   a2.location.name as locationName,
   a2.timestamp as timestamp
from pattern [every (a1 = HomeEvent(a1.tvConsumption >= 20) -> (
    timer:interval(6 hours) and not a2 = HomeEvent((a2.home = a1.home and a2.
    tvConsumption < 20))))]
```

En cuanto al código de las acciones generado por el editor, el Listado 7.8 muestra el código XML con la acción de envío por correo electrónico cuando se detecte un evento complejo de tipo `IrresponsibleTelevisionUse`.

Listado 7.8: Fichero *IrresponsibleTelevisionUse.action* generado por el editor de patrones.

```
<?xml version="1.0" encoding="UTF-8"?>

<mule xmlns:twitter="http://www.mulesoft.org/schema/mule/twitter"
    xmlns:smtps="http://www.mulesoft.org/schema/mule/smtps"
    xmlns:smtp="http://www.mulesoft.org/schema/mule/smtp"
    xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
    xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
    xmlns:spring="http://www.springframework.org/schema/beans" version="EE
        -3.4.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://
        www.springframework.org/schema/beans/spring-beans-current.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/
            schema/mule/core/current/mule.xsd
        http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/
            schema/mule/vm/current/mule-vm.xsd
        http://www.mulesoft.org/schema/mule/smtp http://www.mulesoft.org/
            schema/mule/smtp/current/mule-smtp.xsd
        http://www.mulesoft.org/schema/mule/smtps http://www.mulesoft.org/
            schema/mule/smtps/current/mule-smtps.xsd
        http://www.mulesoft.org/schema/mule/twitter http://www.mulesoft.org/
            schema/mule/twitter/2.4/mule-twitter.xsd">
    <flow name="IrresponsibleTelevisionUse">
        <processor-chain>
            <set-payload value="Detected Alert '#[message.inboundProperties
                ['eventPatternName']]': #[payload]" doc:name="Set Email Body"
                />
            <smtps:outbound-endpoint host="smtp.uca.es" port="465" user="
                ****" password="****" to="juanboubet@hotmail.com" from="juan.
                boubeta@uca.es" cc="" subject="Alert:
                IrresponsibleTelevisionUse" responseTimeout="3000"/>
        </processor-chain>
    </flow>
</mule>
```

7.1.7. Detección y Notificación de Situaciones Críticas o Relevantes

Con la finalidad de que puedan detectarse y notificarse las situaciones críticas o relevantes sobre domótica modeladas gráficamente, es requisito indispensable que el código EPL de los patrones se registre en el motor Esper y, además, el código XML de sus acciones se añada al ESB Mule. Esto se llevará a cabo de forma automática, siempre y cuando el usuario haya seleccionado *new-eventpattern* como directorio de la aplicación Mule donde deben crearse los ficheros EPL y *new-action* como directorio donde deben crearse dichos ficheros XML. Esto será gestionado por los flujos Mule *EventPatternAdditionToEsper* y *ActionForEventPatternAdditionToMule*, respectivamente (véase la Sección 6.4).

Por otra parte, el flujo *ComplexEventReceptionAndDecisionMaking* recibirá todos los eventos complejos detectados encargándose, posteriormente, de realizar todas las acciones añadidas para los patrones que hayan creado estos eventos.

7.2. Caso de Estudio sobre Seguridad en Redes

Las redes de ordenadores se han convertido en elementos críticos de las infraestructuras IT (*Information Technology*) actuales, debido a que la aparición de fallos o los ataques realizados sobre estas redes pueden tener grandes repercusiones tanto a nivel económico como en la propia vida de las personas [Gad12a]. Por ello, es primordial el análisis y la monitorización de estas redes de ordenadores, así como del tráfico que fluye por ellas, con el objetivo de prevenir, o al menos detectar lo antes posible, estas situaciones críticas.

La captura de paquetes de red es uno de los métodos existentes en la actualidad para obtener los datos de redes TCP/IP que podrán ser posteriormente tratados por sistemas específicos como los analizadores de paquetes (o *sniffers*). Sin embargo, existen dos grandes retos a los que los sistemas informáticos, en general, deben enfrentarse: la ingente cantidad de información que deben manipular continuamente y la necesidad de procesarla lo antes posible. Con el propósito de paliar estos problemas, se ha propuesto en [Gad12a] el uso de CEP; gracias a la definición de patrones de eventos podrán detectarse situaciones críticas en el ámbito de la seguridad en redes como pueden ser la congestión del tráfico de red o el ataque de denegación de servicios, entre otros.

Así pues, en este caso de estudio también se hace uso del enfoque propuesto en el Capítulo 3 pero aplicando, esta vez, la propuesta de integración de CEP en SOA 2.0, presentada en el Capítulo 6, al ámbito de la seguridad en redes.

Como productor de eventos se ha escogido un generador de eventos de paquetes de red, desarrollado por Ruediger Gad, dentro del Grupo de Investigación *IT Security, Network Security and Privacy* de la *University of Applied Sciences Frankfurt am Main* (Alemania). Este generador de eventos está escrito en Clojure [Hic14] y utiliza la biblioteca jNetPcap [Tec14] para la captura de los paquetes de red. Fundamentalmente, conforme captura estos paquetes los va filtrando a partir de unas reglas simples definidas internamente como, por ejemplo, si se trata de un paquete TCP y, además, la bandera (*flag*) SYN o FIN

está activada. Entonces, estos paquetes filtrados son enviados a una cola de mensajes; concretamente, se utiliza Apache ActiveMQ [Apa14a], uno de los agentes JMS de código abierto más populares en la actualidad, como infraestructura de comunicación donde el intercambio de mensajes, en esta caso, se realizará utilizando el modelo publicador/suscriptor mediante el uso de los denominados *topics*. De hecho, este generador utiliza un *topic* diferente por cada tipo de evento que manipule. En particular, el *topic* denominado *sniffer.header.parsed* es donde se deposita cada evento proveniente del *sniffer* que contiene los datos de la cabecera del paquete capturado que ha causado este evento y que ha sido transformado a formato *Map<String, Object>*. Como consumidor de eventos se ha establecido únicamente el servicio de correo electrónico.

Posteriormente, se detalla la implementación del flujo Mule que permite obtener y gestionar los eventos desde dicho generador de eventos y su integración con la arquitectura descrita en el Capítulo 6, seguido de los pasos a llevar a cabo, según el enfoque definido en el Capítulo 3, desde que el experto en el dominio define el dominio sobre seguridad en redes hasta que los usuarios reciben las notificaciones en tiempo real de las situaciones acontecidas en las que estén interesados.

7.2.1. Recepción y Gestión de Eventos Provenientes de un Generador de Paquetes de Red

Tal y como se indicó con anterioridad, la arquitectura propuesta en el Capítulo 6 es extensible a distintos dominios de aplicación, requiriendo únicamente la implementación de un nuevo flujo Mule específico que permita conectar dicha arquitectura con el productor de eventos correspondiente. Por este motivo, se ha implementado el flujo denominado *EventReceptionAndManagement-NetworkAnalysis* que permite integrar el generador de eventos de paquetes de red seleccionado en este caso de estudio —el productor de eventos— con Mule (véase la Figura 7.9).

Este flujo está compuesto por un *endpoint* JMS de entrada, un procesador de mensajes *collection splitter* y, finalmente, un componente de referencia al flujo *DomainDynamicGenerationAndEventSendingToEsper*.

El *endpoint* JMS es el responsable de establecer la comunicación entre el generador de eventos y Mule. JMS es una API que hace posible que la comunicación entre distintos componentes de una aplicación distribuida presente un bajo acoplamiento y sea fiable y asíncrona. Esta API soporta dos tipos de modelos de mensajería: *queues*, que son punto a punto, y *topics*, que se basan en el modelo publicador/suscriptor. En un modelo punto a punto, un emisor envía mensajes a una cola específica y un receptor lee los mensajes desde una cola, por lo cual, el emisor conoce el destino del mensaje y envía el mensaje directamente a la cola del receptor. Por otro lado, en el modelo publicador/suscriptor se envían los mensajes a un *topic* de mensajes específico y son los suscriptores los que deben mostrar su interés en recibir los mensajes desde este *topic*, modelo que se caracteriza porque no existe ninguna relación directa entre los productores y los consumidores. La principal ventaja de usar un *topic* estriba en que varios consumidores pueden recibir un mismo

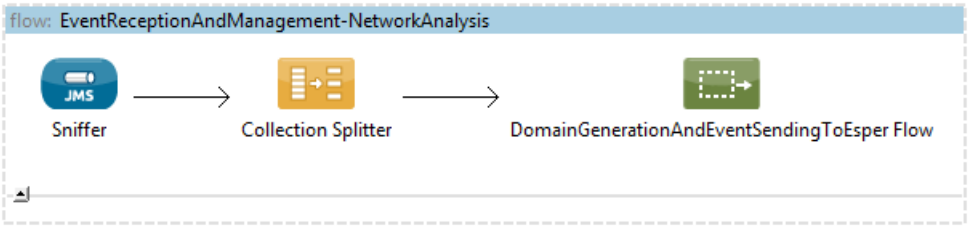

Figura 7.9: Implementación del flujo de recepción y gestión de eventos del caso de estudio sobre seguridad en redes.

mensaje, al contrario que cuando se usa una *queue*, donde únicamente un consumidor obtendrá el mensaje.

Básicamente, se ha configurado dicho *endpoint* especificando que el tipo de agente JMS a utilizar sea Apache ActiveMQ con el *topic sniffer.header.parsed*, precisamente donde este generador depositará los eventos conforme los vaya creando. Gracias a esta sencilla configuración, se ha logrado conectar el generador de eventos con Mule.

Es importante destacar que cada mensaje recibido en Mule desde el generador consiste en una colección formada por varios eventos simples en formato *Map<String, Object>*. Por esta razón, se ha conectado el *endpoint* con el procesador de mensajes *collection splitter*, que se encargará de separar todo el contenido del mensaje en varios eventos simples. La clave de este *map* de tipo *String* registrará el nombre del tipo de evento (*sniffer.header.parsed*) mientras que el valor de tipo *Object* será, a su vez, otro *map* de tipo *Map<String, Object>* que almacenará todas las propiedades de dicho evento, registrándose cada nombre de la propiedad con su respectivo valor.

Debido a que el productor de eventos ya proporciona los datos en formato *Map<String, Object>*, en este caso de estudio, a diferencia del caso de estudio sobre domótica, no ha sido necesaria la implementación de ningún transformador.

Finalmente, el componente de referencia de flujo enviará cada evento simple de tipo *sniffer.header.parsed* al flujo *DomainDynamicGenerationAndEventSendingToEsper*, el cual se encargará de generar el dominio CEP automáticamente así como enviar estos eventos al motor CEP para su procesamiento.

A partir de este momento, la arquitectura ya podrá analizar y procesar información proveniente del generador de eventos de paquetes de red. No obstante, si lo que se pretende es, además, detectar situaciones críticas y/o relevantes para este dominio y notificarlas a los sistemas y usuarios interesados, entonces deberán incorporarse al motor CEP los patrones de eventos para llevar a cabo dicha detección. Seguidamente, se detallan los pasos a seguir para lograrlo.

7.2.2. Definición del Modelo de Dominio CEP

Un experto en el dominio de seguridad en redes será el responsable de definir gráficamente el dominio CEP conforme al metamodelo descrito en la Sección 4.2.1. Este dominio

estará compuesto por los tipos de eventos y propiedades que describan el dominio sobre seguridad en redes; concretamente, el tipo de evento *sniffer.header.parsed* con sus propiedades (véase la Tabla 7.3).

Tabla 7.3: Propiedades del tipo de evento *sniffer.header.parsed*.

Propiedad	**Tipo**	**Descripción**
ts	*Long*	Fecha y hora en las que el paquete se ha capturado.
len	*Long*	La longitud del paquete.
ethSrc	*String*	La dirección Ethernet de origen —dirección MAC (*Media Access Control*).
ethDst	*String*	La dirección Ethernet de destino —dirección MAC.
arpOpDesc	*String*	La descripción de la operación ARP (*Address Resolution Protocol*).
arpTargetMac	*String*	La dirección MAC de destino ARP.
arpTargetIp	*String*	La dirección IP de destino ARP.
arpSourceMac	*String*	La dirección MAC de origen ARP.
arpSourceIp	*String*	La dirección IP de origen ARP.
ipSrc	*String*	La dirección IP de origen.
ipDst	*String*	La dirección IP de destino.
ipVer	*Long*	La versión IP (4 o 6).
ipId	*Long*	El identificador IP.
ipTtl	*Long*	El tiempo de vida IP.
ipChecksum	*Long*	La suma de verificación IP.
icmpEchoSeq	*Long*	El número de secuencia eco ICMP (*Internet Control Message Protocol*).
icmpType	*String*	El tipo de paquete ICMP (petición o respuesta eco).
tcpSrc	*Long*	El puerto TCP de origen.
tcpDst	*Long*	El puerto TCP de destino.
tcpAck	*Long*	El número de acuse de recibo TCP.
tcpSeq	*Long*	El número de secuencia TCP.
tcpFlags	*Long*	Las banderas TCP: SYN, ACK...
tcpTsval	*Long*	Fecha y hora TCP.
tcpTsecr	*Long*	La respuesta eco de fecha y hora TCP.
udpDst	*Long*	El puerto UDP (*User Datagram Protocol*) de destino.
udpSrc	*Long*	El puerto UDP de origen.

Al igual que se ha comentado para el caso de estudio sobre domótica, esta tarea podrá ser realizada por el experto en el dominio bien utilizando el editor gráfico de dominios CEP (véase el Capítulo 4) o bien directamente mediante el editor gráfico de patrones de eventos (véase el Capítulo 5), ya que este editor, además de ofrecer sus funcionalidades propias de gestión de patrones, también permite la definición de un modelo de dominio

CEP, siempre y cuando el editor todavía no haya sido reconfigurado para un dominio en particular. En esta ocasión, se ha optado por utilizar directamente el editor gráfico de patrones de eventos; por consiguiente, una vez diseñado, validado y guardado el dominio de seguridad, este editor se reconfigurará automáticamente sin requerir ninguna operación adicional.

Del mismo modo a como se procedería con el editor de dominios, el dominio CEP puede crearse de dos formas distintas utilizando el editor de patrones: manualmente —opción del menú *CEP Domain > New...*— lo que implica que el experto debe definir el tipo de evento *sniffer.header.parsed* y sus propiedades haciendo uso de las herramientas *Event* y *EventProperty* disponibles en la paleta del editor; dinámicamente —opción del menú *CEP Domain > Auto-detect Domain*— que generará automáticamente el dominio gráfico a partir del tipo de eventos inferido desde los eventos que viajen por el flujo *DomainDynamicGenerationAndEventSendingToEsper* del ESB.

Se ha optado por la generación automática del dominio, realizándose seguidamente algunas modificaciones sobre el mismo. En concreto, se ha asignado el nombre *network-analysis* al dominio formado únicamente por el tipo de eventos inferido —*sniffer.header.parsed*—, así como una descripción textual. Además, se ha asociado este tipo de eventos con un icono gráfico para mejorar la usabilidad, de modo que cualquier usuario pueda reconocerlo de una forma intuitiva. La Figura 7.10 ilustra el dominio CEP modelado sobre seguridad; téngase en cuenta que no se muestra la imagen completa del tipo de evento por razones de espacio. Todas sus propiedades se encuentran definidas en la Tabla 7.3. Como puede observarse, algunas de estas propiedades han sido reordenadas en el modelo gráfico para situar en las primeras posiciones aquellas que se referenciarán en los patrones de eventos propuestos.

7.2.3. Validación y Almacenamiento del Modelo de Dominio CEP

Posteriormente al diseño del dominio de seguridad, se ha validado este automáticamente a partir de las restricciones definidas en la Sección 4.2.1, y se ha guardado en el sistema tras comprobarse que el modelo es correcto y que no incumple ninguna de las restricciones. Para ejecutar estas funcionalidades se ha seleccionado la opción del menú *CEP Domain > Save and Validate* del editor gráfico de patrones.

Una vez diseñado, validado y guardado dicho dominio, este editor se reconfigurará automáticamente sin requerir ninguna operación adicional. A partir de este momento, el usuario que así lo requiera podrá proceder al modelado de patrones de eventos para el ámbito de la seguridad en redes.

7.2.4. Definición de Modelos de Patrón de Eventos

Para que la definición gráfica de patrones de eventos sobre seguridad en redes pueda llevarse a cabo, es necesario que previamente se haya reconfigurado el editor gráfico de

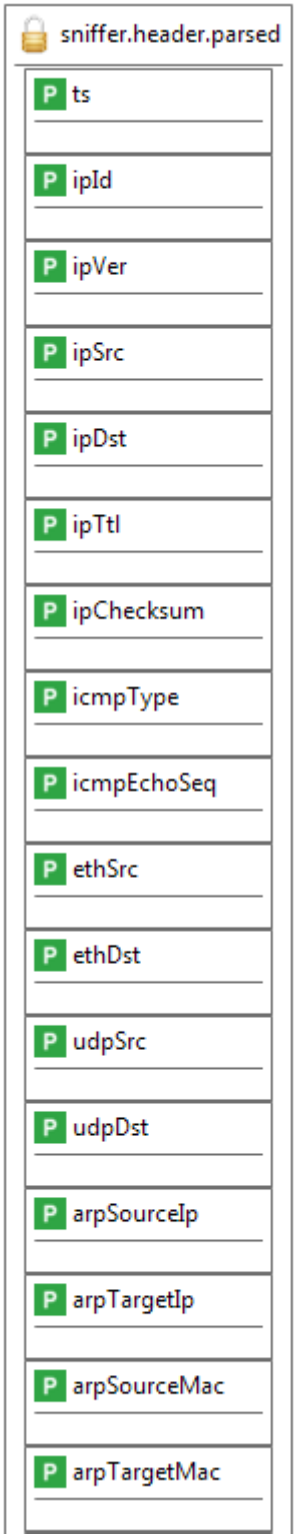

Figura 7.10: Dominio CEP de seguridad en redes.

patrones con el dominio *network-analysis*. Esto provocará que se añada automáticamente el tipo de eventos *sniffer.header.parsed* como una herramienta más en la paleta del editor, posicionándose en el grupo de herramientas denominado *Simple Events* (véase la Figura 7.11). Por lo tanto, cuando el usuario necesite describir alguna condición de patrón donde *sniffer.header.parsed* deba estar presente, hará uso de dicha herramienta para añadir este tipo de eventos en el canvas del editor.

Como se ha comentado previamente, gracias a esta personalización dinámica de la paleta del editor, se aumenta considerablemente la usabilidad y experiencia del usuario final al no tener que preocuparse de cómo representar el tipo de eventos y sus propiedades; simplemente deberá usarlo tal cual sea visualizado en el canvas. Por otro lado, esto proporciona otra ventaja fundamental: el usuario final no puede realizar ninguna modificación sobre el tipo de eventos dado, ya que esto conllevaría a la creación de patrones de eventos inconsistentes, al usarse definiciones distintas para un mismo tipo de eventos.

A continuación, se detallan los patrones de eventos definidos gráficamente en este caso de estudio para la detección de situaciones relevantes en el ámbito de la seguridad en redes, más específicamente, *icmp echo request*, *icmp echo reply* e *icmp ping reponse time*.

Patrón de *ICMP Echo Request*

A continuación, se describe el patrón de *icmp_echo_request* que se pretende modelar con el editor gráfico de patrones de eventos:

- **Nombre**: *icmp_echo_request*.
- **Descripción**: Este patrón permite detectar un mensaje de control que se envía a un *host* con el propósito de recibir de él una respuesta eco (o mensaje *echo-reply*).
- **Condiciones del patrón**:
  - El valor de la propiedad *icmpType* de un evento *sniffer.header.parsed* debe ser igual al valor *echo request*.
- **Nuevo evento complejo**: *icmp_echo_request*. Se creará un nuevo evento complejo con las siguientes propiedades:
  - *timestamp*: la fecha y la hora en la que se ha producido el evento *sniffer.header.parsed* seleccionado (propiedad *ts*), medido en nanosegundos.
  - *source*: el mismo valor de la propiedad *ipSrc* del evento *sniffer.header.parsed*.
  - *destination*: el mismo valor de la propiedad *ipDst* del evento *sniffer.header.parsed*.
- **Acciones:**

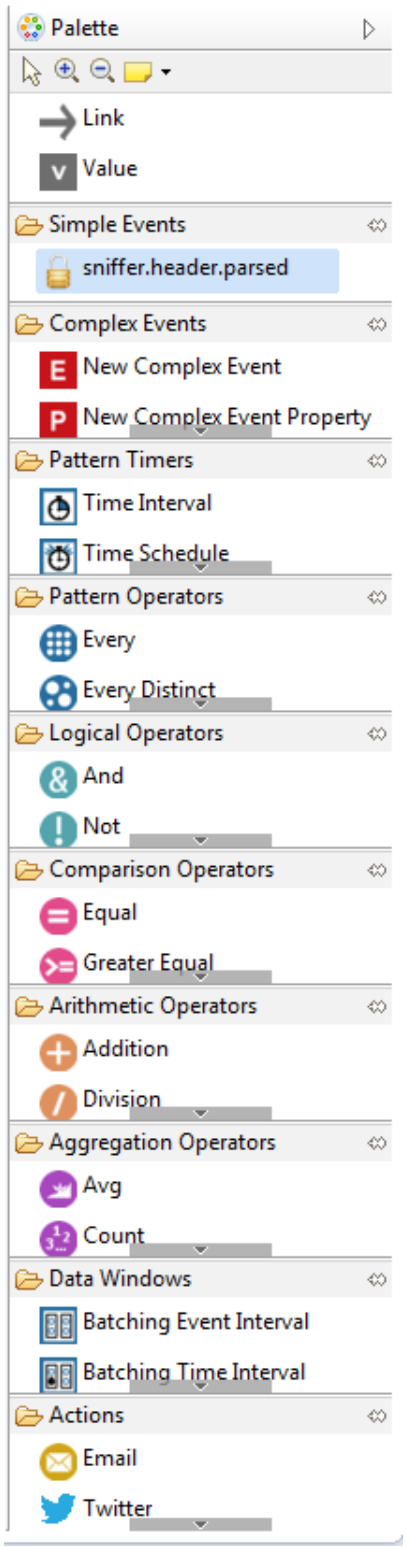

Figura 7.11: Paleta del editor reconfigurable personalizada con el dominio de seguridad.

- *Email*: cada nuevo evento complejo de tipo *icmp_echo_request* creado, tras el cumplimiento de las condiciones del patrón, será enviado a la dirección o direcciones de correo electrónico especificadas en el componente *Email* (atributo *To*). Para ello, deberá indicarse también información sobre la dirección del emisor del mensaje, el *host* y el puerto, el nombre de usuario y contraseña, así como el asunto del correo, por ejemplo, *Alert: ICMP Echo Request*.

La Figura 7.12 presenta el modelo de este patrón diseñado con el editor de patrones. Como puede observarse, se ha asignado una imagen al icono que representará al nuevo tipo de evento complejo *icmp_echo_request*. El evento *sniffer.header.parsed* no se muestra al completo por razones de espacio.

Patrón de *ICMP Echo Reply*

Seguidamente, se describe el patrón de *icmp_echo_reply* que se pretende modelar con el editor gráfico de patrones de eventos:

- **Nombre**: *icmp_echo_reply*.
- **Descripción**: Este patrón permite detectar un mensaje de control generado como respuesta a una petición eco (o mensaje *echo-request*).
- **Condiciones del patrón**:
  - El valor de la propiedad *icmpType* de un evento *sniffer.header.parsed* debe ser igual al valor *echo reply*.
- **Nuevo evento complejo**: *icmp_echo_reply*. Se creará un nuevo evento complejo con las siguientes propiedades:
  - *timestamp*: la fecha y la hora en la que se ha producido el evento *sniffer.header.parsed* seleccionado (propiedad *ts*), medido en nanosegundos.
  - *source*: el mismo valor de la propiedad *ipSrc* del evento *sniffer.header.parsed*.
  - *destination*: el mismo valor de la propiedad *ipDst* del evento *sniffer.header.parsed*.
- **Acciones:**
  - *Email*: cada nuevo evento complejo de tipo *icmp_echo_reply* creado será enviado a la dirección o direcciones de correo electrónico especificadas en el componente *Email*.

La Figura 7.13 ilustra el modelo de este patrón diseñado con el editor de patrones. Como puede observarse, se ha asignado una imagen al icono que representará al nuevo tipo de evento complejo *icmp_echo_reply*. El evento *sniffer.header.parsed* no se muestra al completo por razones de espacio.

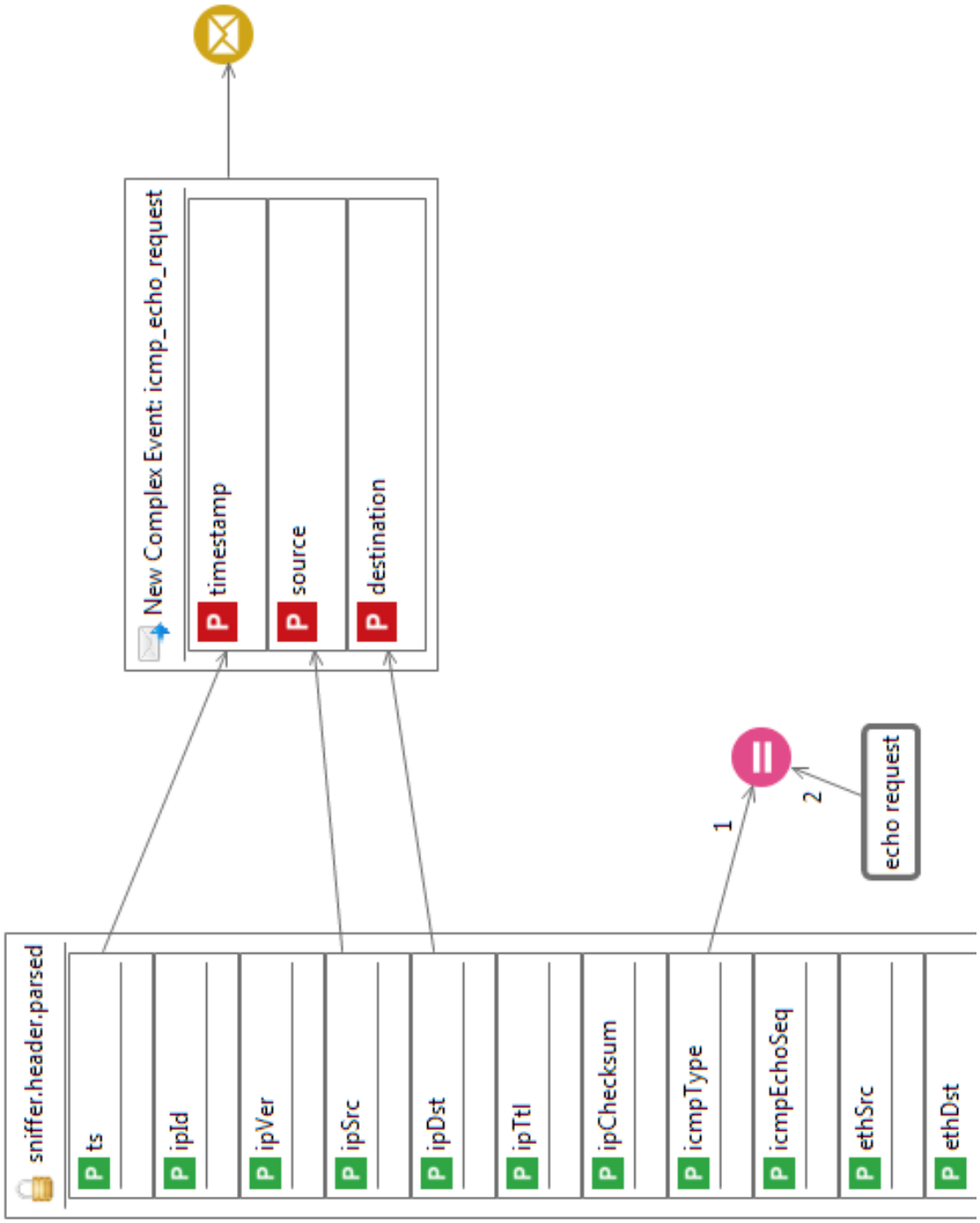

Figura 7.12: Modelo del patrón *icmp_echo_request*.

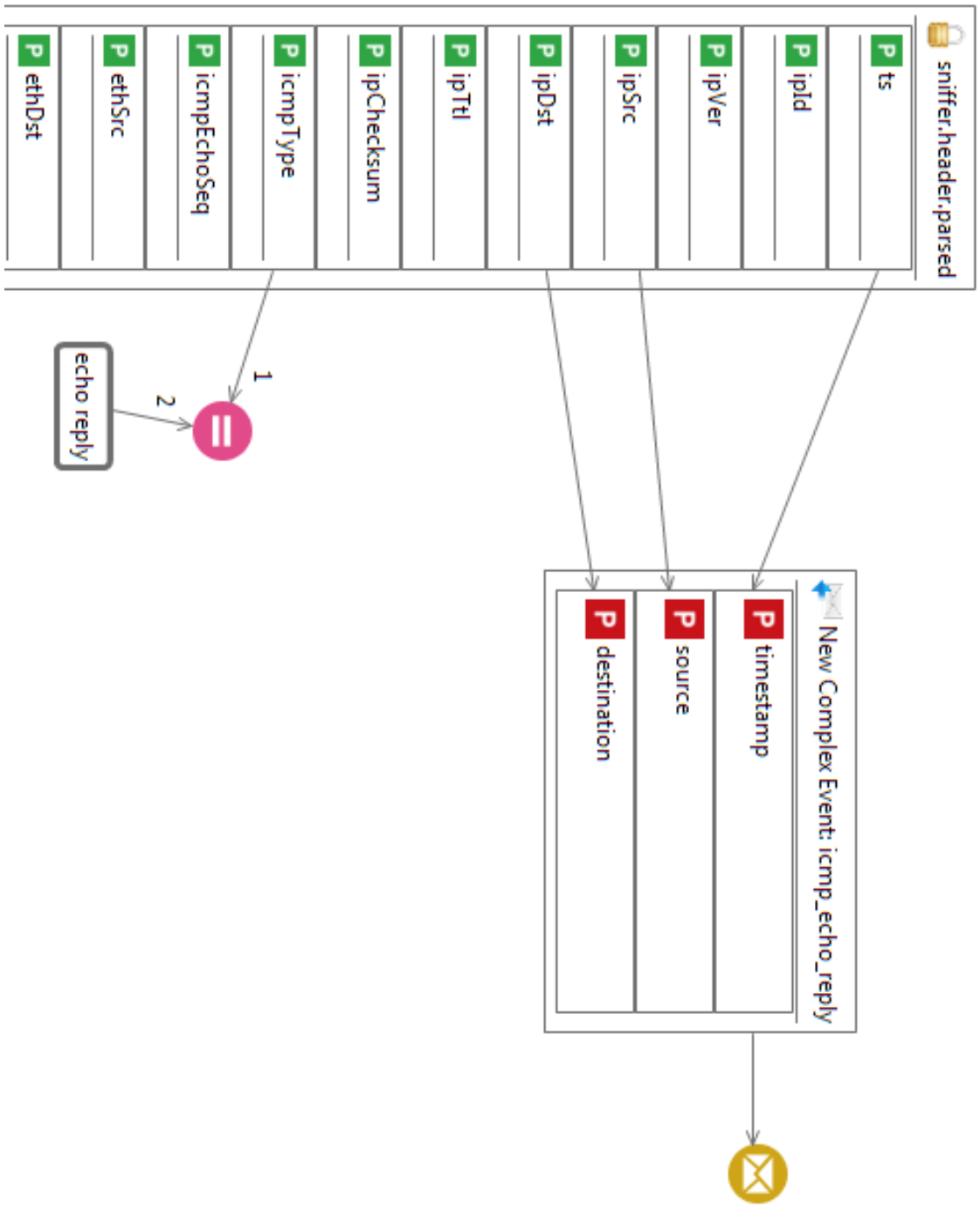

Figura 7.13: Modelo del patrón *icmp_echo_reply*.

Patrón de *ICMP Ping Response Time*

A continuación, se describe el patrón de *icmp_ping_response_time* que se pretende modelar con el editor gráfico de patrones de eventos:

- **Nombre**: *icmp_ping_response_time*.
- **Descripción**: Este patrón permite calcular la diferencia temporal entre la ocurrencia de un mensaje *ICMP echo request* y un mensaje *ICMP echo reply*.
- **Condiciones del patrón**:
  - A cada evento complejo *icmp_echo_request* (generado por el patrón de *icmp_echo_request*) le sigue temporalmente un evento complejo *icmp_echo_reply* (generado por el patrón de *icmp_echo_reply*).
  - Adicionalmente, se cumple que el valor de la propiedad *source* del evento complejo *icmp_echo_reply* es igual al de la propiedad *destination* del evento complejo *icmp_echo_request* y, además, el valor de la propiedad *destination* del evento *icmp_echo_reply* es igual al de la propiedad *source* del evento *icmp_echo_request*.
- **Nuevo evento complejo**: *icmp_ping_response_time*. Se creará un nuevo evento complejo con las siguientes propiedades:
  - *requestTimestamp*: el valor de la propiedad *timestamp* del evento complejo *icmp_echo_request*.
  - *responseTimestamp*: el valor de la propiedad *timestamp* del evento complejo *icmp_echo_reply*.
  - *time*: la diferencia del *timestamp* del evento complejo *icmp_echo_reply* menos el *timestamp* del evento complejo *icmp_echo_request*, esto es, la diferencia del tiempo en el que se ha detectado el evento *icmp_echo_reply* menos el tiempo en el que se ha detectado el evento *icmp_echo_request*.
  - *source*: el valor de la propiedad *source* del evento complejo *icmp_echo_request*.
  - *destination*: el valor de la propiedad *destination* del evento complejo *icmp_echo_request*.
- **Acciones:**
  - *Email*: cada nuevo evento complejo de tipo *icmp_ping_response_time* creado será enviado a la dirección o direcciones de correo electrónico especificadas en el componente *Email*.

La Figura 7.14 presenta el modelo de este patrón diseñado con el editor de patrones. Como puede observarse, se ha asignado una imagen al icono que representará al nuevo tipo de evento complejo *icmp_ping_response_time*.

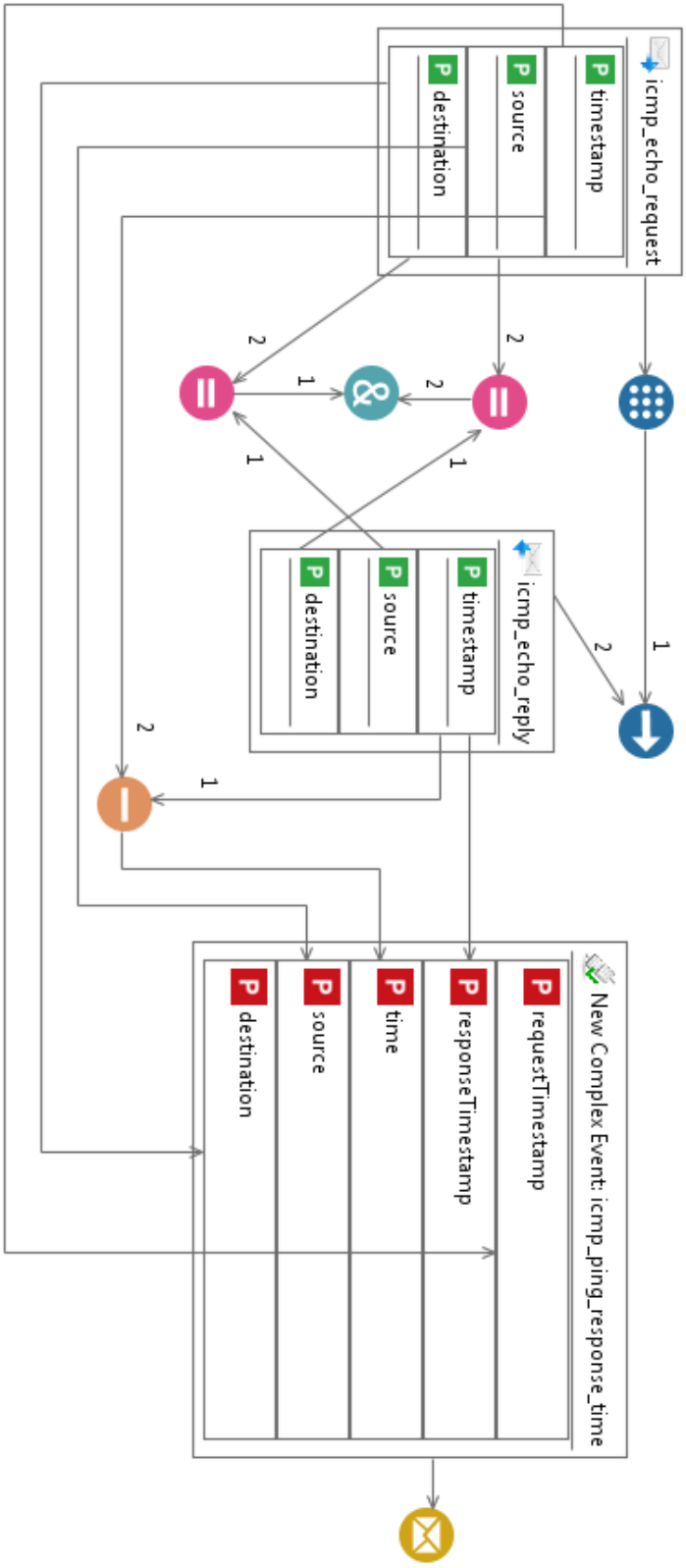

Figura 7.14: Modelo del patrón *icmp_ping_response_time*.

7.2.5. Validación y Almacenamiento de Modelos de Patrón de Eventos

Conforme se ha ido modelando cada patrón de seguridad detallado anteriormente, se ha validado automáticamente a partir de las restricciones definidas en la Sección 5.2.1, y se ha guardado en el sistema tras comprobarse que el modelo es correcto y que no incumple ninguna de las restricciones. Para ejecutar estas funcionalidades se ha seleccionado la opción del menú *Event Patterns > Save and Validate* del editor gráfico de patrones.

Tras el almacenamiento de cada modelo de patrón en el sistema, se ha reconfigurado automáticamente la paleta del editor, añadiéndose el nuevo tipo de evento complejo definido en cada modelo como una herramienta más dentro del grupo de herramientas denominado *Complex Events* (véase la Figura 7.15). Por lo tanto, cuando el usuario necesite describir alguna condición de patrón en la cual deba estar presente alguno de los tipos de eventos complejos diseñados previamente, hará uso de las herramientas de eventos complejos de la paleta del editor para añadir estos tipos de eventos en el canvas del editor. De esta forma es precisamente como se ha definido el modelo del patrón de *icmp_ping_response_time* (véase la Figura 7.14): los tipos de eventos complejos *icmp_echo_request* y *icmp_echo_reply*, junto con sus propiedades, requeridos para definir el patrón de *icmp_ping_response_time* han sido añadidos al modelo utilizando sus herramientas correspondientes de la paleta del editor.

El hecho de que la paleta del editor se reconfigure automáticamente facilita al usuario la descripción de patrones de eventos de una forma amigable, así como el diseño de jerarquías de patrones de eventos, como se ha demostrado durante el modelado de los patrones de seguridad.

Posteriormente, se ha procedido a la exportación de estos patrones de eventos. Para ello, se ha utilizado la opción del menú *File > Export Event Patterns*, obteniéndose un archivo ZIP, denominado *network-analysis_patterns.zip*, que contiene tanto los modelos de los patrones de eventos (modelos EMF) y sus ficheros de diagrama junto con las imágenes a las que hagan referencia dichos modelos de patrón, como el modelo de dominio (modelo EMF) para el que se ha definido estos patrones, su fichero de diagrama y las imágenes a las que haga referencia dicho modelo de dominio.

7.2.6. Transformación de Modelos de Patrón de Eventos a Código

Tras el diseño, la validación y el almacenamiento de los patrones de eventos podrá generarse de forma automática tanto el código EPL de los patrones que deben ser añadidos en tiempo de ejecución en el motor Esper, como el código XML correspondiente a las acciones que deben llevarse a cabo en el ESB Mule cuando se detecten dichos patrones. Para lograrlo, se seleccionará la opción del menú del editor *Event Patterns > Generate and Deploy Pattern Code*.

Al utilizar esta opción, se pedirá al usuario que elija el directorio donde desea que el editor cree los ficheros con extensión *.epl* con el código de los patrones, y el directorio donde desea que se registren los ficheros *.action* con las acciones de los patrones.

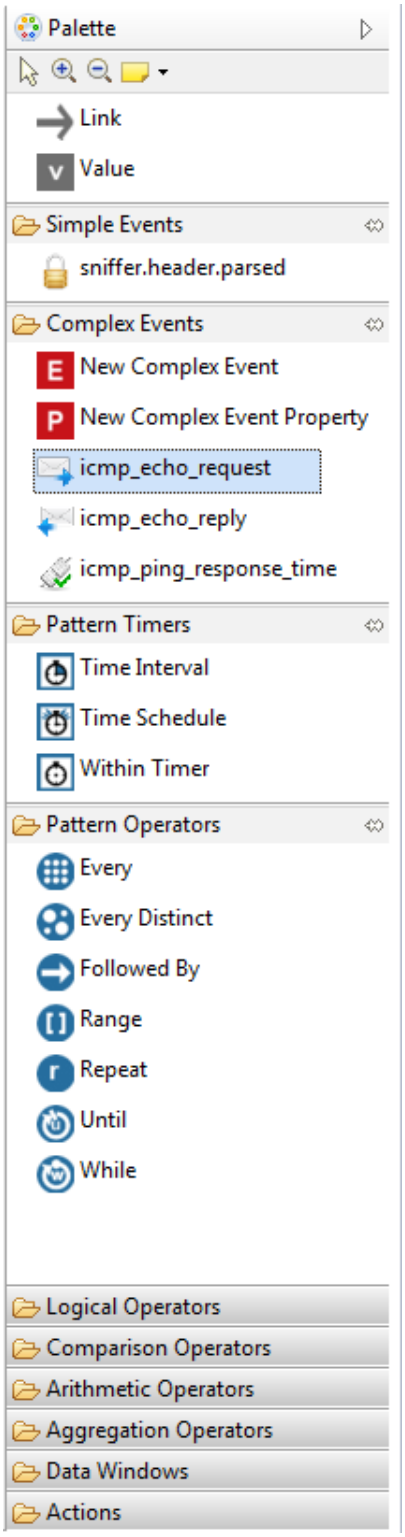

Figura 7.15: Paleta del editor reconfigurable personalizada con los eventos complejos de seguridad.

A continuación, se especifican los ficheros *.epl* y *.action* generados automáticamente para cada uno de los patrones de seguridad definidos y modelados en la Sección 7.2.4.

Patrón de *ICMP Echo Request*

Este patrón permite detectar un mensaje de control ICMP que se envía a un *host* con el propósito de recibir de él una respuesta eco (o mensaje *echo-reply*). El Listado 7.9 muestra el código EPL generado para este patrón.

Listado 7.9: Fichero *icmp_echo_request.epl* generado por el editor de patrones.

```
@Name('icmp_echo_request')
insert into icmp_echo_request
select a1.ts as timestamp,
   a1.ipSrc as source,
   a1.ipDst as destination
from sniffer.header.parsed a1
where a1.icmpType = 'echo request'
```

A diferencia de los patrones de eventos de domótica, en este patrón no se ha generado la cláusula `from pattern`. Esto es debido a que no se ha incluido ningún operador ni temporizador de patrón en el modelo. En su lugar se ha utilizado la cláusula `from` donde se indica para qué tipo de evento —*sniffer.header.parsed*— se desea establecer las condiciones, asociándosele el alias `a1`. La condición a determinar en este patrón —el valor de la propiedad *icmpType* de un evento *sniffer.header.parsed* debe ser igual al valor *echo request*— se ha especificado haciendo uso de la cláusula `where`.

Finalmente, se seleccionan las siguientes propiedades: el tiempo de registro, la IP origen y la IP destino. Con estas propiedades se crea un nuevo evento complejo de tipo `icmp_echo_request(timestamp, source, destination)` y se inserta en un nuevo flujo de eventos de Esper con la misma denominación que el evento complejo.

Por otra parte, el editor de patrones ha creado el fichero *icmp_echo_request.action* con el código XML correspondiente a las acciones definidas gráficamente para este patrón. Cabe destacar que este fichero es idéntico al fichero *PowerFailure.action* (véase el Listado 7.6), excepto en el nombre del flujo, que se denomina *icmp_echo_request*, y las direcciones de correo a las que se desean enviar los eventos complejos *icmp_echo_request* detectados.

Patrón de *ICMP Echo Reply*

Este patrón permite detectar un mensaje de control generado como respuesta a una petición eco (o mensaje *echo-request*). El Listado 7.10 muestra el código EPL generado para este patrón.

Este código es muy similar al del patrón de *icmp_echo_request*, excepto en la condición establecida: el valor de la propiedad *icmpType* del evento *sniffer.header.parsed* debe ser igual al valor *echo reply*. El tipo de evento complejo que creará este patrón es el siguiente: `icmp_echo_reply(timestamp, source, destination)`.

Listado 7.10: Fichero *icmp_echo_reply.epl* generado por el editor de patrones.

```
@Name('icmp_echo_reply')
insert into icmp_echo_reply
select a1.ts as timestamp,
   a1.ipSrc as source,
   a1.ipDst as destination
from sniffer.header.parsed a1
where a1.icmpType = 'echo reply'
```

Adicionalmente, el editor de patrones ha creado el fichero *icmp_echo_reply.action* con el código XML correspondiente a las acciones definidas gráficamente para este patrón. Este fichero es también idéntico al fichero *PowerFailure.action* (véase el Listado 7.6), excepto en el nombre del flujo, que se denomina *icmp_echo_reply*, y las direcciones de correo a las que se desean enviar los eventos complejos *icmp_echo_reply* detectados.

Patrón de *ICMP Ping Response Time*

Este patrón permite calcular la diferencia temporal entre la ocurrencia de un mensaje *ICMP echo request* y un mensaje *ICMP echo reply*. El Listado 7.11 muestra el código EPL generado para este patrón.

Listado 7.11: Fichero *icmp_ping_response_time.epl* generado por el editor de patrones.

```
@Name('icmp_ping_response_time')
insert into icmp_ping_response_time
select a1.timestamp as requestTimestamp,
   a2.timestamp as responseTimestamp,
   (a2.timestamp - a1.timestamp) as time,
   a1.source as source,
   a1.destination as destination
from pattern [(every a1 = icmp_echo_request -> a2 = icmp_echo_reply((a2.
   source = a1.destination and a2.destination = a1.source)))]
```

En este patrón sí se han usado operadores de patrón; por consiguiente, se ha generado la cláusula `from pattern`. Aquí es donde se determinan las condiciones del patrón: para cada (`every`) evento complejo de tipo *icmp_echo_request* —al que se le ha asignado el alias `a1`— debe seguirle temporalmente un evento complejo de tipo *icmp_echo_reply* —al que se le ha asignado el alias `a2`— en el que el valor de su propiedad *source* sea igual al de la propiedad *destination* del evento *icmp_echo_request* y, además, el valor de su propiedad *destination* sea igual al de la propiedad *source* del evento *icmp_echo_request*.

Si se cumplen estas condiciones, entonces se creará un nuevo evento complejo de tipo `icmp_ping_response_time(requestTimestamp, responseTimestamp, time, source, destination)` en el que la propiedad `requestTimestamp` coincide con el valor del tiempo de registro del evento *icmp_echo_request*, la propiedad `responseTimestamp` es igual al tiempo de registro del evento *icmp_echo_reply*, la propiedad `time` es el resultado de calcular la diferencia del tiempo de registro del evento *icmp_echo_reply* menos el tiempo de registro del evento *icmp_echo_request*, y las propiedades `source` y `destination` coinciden

con las mismas propiedades del evento *icmp_echo_request*. Este evento complejo creado se insertará en un nuevo flujo de eventos de Esper con el mismo nombre: *icmp_ping_response_time*.

Finalmente, el editor de patrones ha creado el fichero *icmp_ping_response_time.action* con el código XML correspondiente a las acciones definidas gráficamente para este patrón. El código incluido en este fichero es similar al incluido en el fichero *PowerFailure.action* (véase el Listado 7.6), excepto en el nombre del flujo, que se denomina *icmp_ping_response_time*, y las direcciones de correo a las que se desean enviar los eventos complejos *icmp_ping_response_time* detectados.

7.2.7. Detección y Notificación de Situaciones Críticas o Relevantes

Con el objeto de que puedan detectarse y notificarse las situaciones críticas o relevantes sobre seguridad modeladas gráficamente, es requisito indispensable que el código EPL de los patrones se registre en el motor Esper y, además, el código XML de sus acciones se añada al ESB Mule. Esto se llevará a cabo de forma automática, siempre y cuando el usuario haya seleccionado *new-eventpattern* como directorio de la aplicación Mule donde deben crearse los ficheros EPL y *new-action* como directorio donde deben crearse dichos ficheros XML. Esto será gestionado por los flujos Mule *EventPatternAdditionToEsper* y *ActionForEventPatternAdditionToMule*, respectivamente (véase la Sección 6.4).

Por otro lado, el flujo *ComplexEventReceptionAndDecisionMaking* recibirá todos los eventos complejos detectados encargándose, posteriormente, de ejecutar todas las acciones añadidas para los patrones que hayan creado estos eventos.

7.3. Conclusiones

En este capítulo se han desarrollado dos casos de estudio en los que se ha aplicado el enfoque dirigido por modelos para el procesamiento de eventos complejos en SOA 2.0, propuesto en esta tesis doctoral. Mientras que el primer caso de estudio trata de detectar situaciones críticas en el ámbito de la domótica, el segundo se centra en el ámbito de la seguridad en redes.

En cuanto al caso de estudio de domótica se ha utilizado la plataforma IoT Xively para la obtención de datos reales provenientes de sensores localizados en tres hogares ubicados en distintas zonas geográficas, a modo de ejemplo. No obstante, podrían usarse los datos de otros hogares e incluso integrarse la arquitectura con otras plataformas IoT como ThingSpeak [Thi14] y con sensores propios.

En el segundo caso de estudio, los datos a ser analizados y procesados provienen de un generador de eventos desarrollado dentro del Grupo de Investigación *IT Security, Network Security and Privacy* de la *University of Applied Sciences Frankfurt am Main* (Alemania) con el que colabora el Grupo de Investigación *UCASE de Ingeniería del Software* de la

UCA, al que pertenece el doctorando. No obstante, si fuese necesario, se podría utilizar otro tipo de productores de eventos en distintos ámbitos.

Es importante señalar que los patrones de eventos propuestos y modelados en este capítulo representan únicamente un subconjunto de los que podrían definirse para cada uno de los dominios tratados. No obstante, suponen una muestra representativa para probar el enfoque.

Del mismo modo que se han utilizado servicios de correo electrónico y de redes sociales como consumidores de eventos, sería viable la integración de la arquitectura con otros consumidores como, por ejemplo, actuadores.

Entre las conclusiones más relevantes que pueden desprenderse de los casos presentados se incluyen las siguientes. Por un lado, el enfoque propuesto es extensible a distintos dominios de aplicación, por lo cual, podrá aplicarse en un futuro próximo a otros ámbitos en los que CEP puede suponer un valor añadido, como pueden ser la fabricación (*manufacturing*) o la salud.

Por otro lado, el DSML para la definición de dominios CEP y el DSML para la definición de patrones de eventos propuestos, así como los editores de dominio y de patrones implementados, abstraen a los usuarios finales de todos los detalles técnicos de implementación referentes a CEP y SOA 2.0, aumentando la usabilidad y la experiencia del usuario. Esto se traduce en poner, al alcance de cualquier persona sin el conocimiento tecnológico, no solo la definición, sino también la notificación en tiempo real, de las situaciones críticas y relevantes sobre el dominio en el que esté interesada.

Capítulo 8

Evaluación

«Para demostrar algo hay que servirse de experimentos y reflexiones y no de autoridad.»
(Paracelso)

En el Capítulo 3 se ha presentado un enfoque dirigido por modelos para el procesamiento de eventos complejos en SOA 2.0. Seguidamente, en los Capítulos 4 y 5 se han definido unos DSML para la definición de dominios CEP y patrones de eventos, y se han desarrollado tanto un editor gráfico para la definición de dominios CEP como otro para la definición de patrones de eventos y su generación a código. Estos editores se utilizan en la solución propuesta para la integración de CEP en SOA 2.0 descrita e implementada en el Capítulo 6.

Así pues, en el presente capítulo se realiza una evaluación tanto de dichos DSML y editores gráficos como del enfoque dirigido por modelos propuestos en esta tesis doctoral.

8.1. DSML para la Definición de Patrones de Eventos

En esta sección se evalúan el DSML para la definición de dominios CEP y el DSML para la definición de patrones de eventos propuestos en los Capítulos 4 y 5, respectivamente. Debido a que el metamodelo de patrones de eventos (véase la Sección 5.2.1) engloba también el metamodelo de dominios CEP (véase la Sección 4.2.1), esto es, el metamodelo de patrones de eventos también posee las mismas metaclases *Event* y *EventProperty* del metamodelo de dominio CEP, se realizará la evaluación directamente sobre el DSML para la definición de patrones de eventos.

Lo que se persigue con este análisis es demostrar que los modelos, que son conformes al metamodelo de patrones de eventos y creados con el editor gráfico de patrones, pueden ser transformados a código implementado en distintos EPL, y no solamente transformados a código EPL de Esper; ya que el enfoque dirigido por modelos propuesto en esta tesis doctoral contempla la posibilidad de añadir otros motores CEP, aparte de Esper. Esta posibilidad de reutilizar los mismos modelos para generar código para distintos lenguajes o plataformas es una de las ventajas clave del desarrollo dirigido por modelos. Conviene

señalar que se ha optado por la utilización del EPL de Esper para probar este enfoque porque es uno de los EPL que ofrecen más operadores de patrón y uno de los más utilizados en la actualidad.

Para conseguir dicho propósito, se ha realizado un análisis comparativo (véase la Tabla 8.1) en el que para cada metaclase de dicho metamodelo se especifica cuál sería su transformación a código EPL de Esper [Esp14], EPL de Oracle [Ora14], StreamSQL [Str14] y CCL [Syb14], algunos de los EPL más conocidos en la actualidad. Téngase en cuenta que para los operadores en los que se ha especificado el tiempo en segundos (*seconds*), podría haberse indicado en otras unidades de tiempo, mientras que en los casos en los que no se ha especificado explícitamente son tratados en segundos. A continuación, se detallan los aspectos más significativos de esta comparativa.

Tabla 8.1: Comparativa entre las metaclases del metamodelo de patrones de eventos y sus equivalencias en código EPL de Esper, EPL de Oracle, StreamSQL y CCL.

Metaclase	**EPL Esper**	**EPL Oracle**	**StreamSQL**	**CCL**
EventPattern	`from pattern`	`MATCHING`	`FROM PATTERN`	`MATCHING`
Condition	`from`	`FROM`	`FROM`	`FROM`
	`where`	`WHERE`	`WHERE`	`WHERE`
ComplexEvent	`insert into`	`INSERT INTO`	`SELECT`	`INSERT INTO`
	`select`	`SELECT`	`INTO`	`SELECT`
ComplexEvent	propiedad	propiedad	propiedad	propiedad
Property	`as` alias	`AS` alias	`AS` alias	`AS` alias
TimeInterval	`timer:interval`	`time_interval`	`interval(n)`	`n seconds`
	`(n seconds)`	`(n seconds)`		
TimeSchedule	`timer:at`		`time(n)`	`AT n`
	`(*,*,*,*,*,*)`			
WithinTimer	`timer:within`	`WITHIN`		
	`(n seconds)`	`n SECONDS`		
Event	evento	evento	evento	evento
EventProperty	propiedad	propiedad	propiedad	propiedad
Value	'cadena'	'cadena'	'cadena'	'cadena'
	número	número	número	número
	`true` o `false`	`true` o `false`	`TRUE` o `FALSE`	`TRUE` o `FALSE`
Every	`every`	`EVERY`		`EVERY`
EveryDistinct	`every-distinct`			`DISTINCT ROWS`
FollowedBy	`->`	`FOLLOWED BY`	`->` y `THEN`	`,`
Range	`[a:b]`	`BETWEEN a`	`BETWEEN a`	`BETWEEN a`
		`AND b`	`AND b`	`AND b`
Repeat	`[n]`	`BETWEEN 1`	`BETWEEN 1`	`n ROWS`
		`AND n`	`AND n`	
Until	`until`			`UNTIL`

(Continúa en la página siguiente)

(Continúa de la página anterior)

Metaclase	EPL Esper	EPL Oracle	StreamSQL	CCL
While	`while`		`FOREACH`	`FOR`
And	`and` y `,`	`AND`	`AND` y `&&`	`AND` y `&&`
Or	`or`	`OR`	`OR` y `||`	`OR` y `||`
Not	`not`	`NOT`	`NOT` y `!`	`NOT` y `!`
Equal	=	=	==	=
GreaterEqual	>=	>=	>=	>=
GreaterThan	>	>	>	>
LessEqual	<=	<=	<=	<=
LessThan	<	<	<	<
NotEqual	!=	!=	!=	!= y <>
Addition	+	+	+	+
Division	/	/	/	/
Modulus	%	%	%	*mod*
Multiplication	*	*	*	*
Subtraction	−	−	−	−
Avg	`avg`	`AVG`	`avg`	`AVG`
Count	`count`	`COUNT`	`count`	`COUNT`
Max	`max`	`MAX`	`max`	`MAX`
Min	`min`	`MIN`	`min`	`MIN`
Sum	`sum`	`SUM`	`sum`	`SUM`
Batching TimeInterval	`win:time_batch (n seconds)`	`RETAIN BATCH OF n SECONDS`	`WITHIN n TIME`	`KEEP EVERY n SECONDS`
Batching EventInterval	`win:length_ batch(n)`	`RETAIN BATCH OF n EVENTS`	`WITHIN n ON id`	`KEEP EVERY n ROWS`
Sliding TimeInterval	`win:time (n seconds)`	`RETAIN n SECONDS`	`SIZE n ADVANCE n TIME`	`KEEP n SECONDS`
Sliding EventInterval	`win:length (n)`	`RETAIN n EVENTS`	`SIZE n ADVANCE n TUPLES`	`KEEP n ROWS`

En primer lugar, la metaclase *EventPatternCondition* es la que representa las condiciones que deben cumplirse para detectar una situación crítica o relevante. Debe tenerse en cuenta que con la finalidad de abstraer al usuario final de la sintaxis particular de un EPL, esta metaclase será transformada bien a una cláusula de tipo búsqueda —`from... where`, en EPL de Esper— utilizada normalmente por este tipo de lenguajes cuando no se ha incluido en las condiciones ningún operador ni operando de tipo patrón, o bien una cláusula de tipo patrón —`from pattern`, en EPL de Esper— cuando las condiciones sí incluyen algún operador u operando específico de patrones. Para lograrlo, se ha tenido que diferenciar estas dos alternativas durante la implementación de las reglas de transformación de los modelos de patrones a código EPL de Esper; de forma que esto sea totalmente transparente al usuario final.

Las metaclases *ComplexEvent* y *ComplexEventProperty*, que describen el tipo de evento complejo a crear cada vez que se detecte el patrón junto con sus propiedades, están relacionadas directamente con la cláusula que se encarga de crear los eventos complejos de este tipo e insertarlos en un flujo específico para ellos —`insert into... select` propiedad `as` alias..., en EPL de Esper.

En cuanto a los operandos de patrón definidos en el metamodelo —*TimeInterval*, *TimeSchedule, WithinTimer* y *Event*—, mencionar que EPL de Oracle no dispone de ninguno equivalente a *TimeSchedule* y que tampoco existen equivalentes a *WithinTimer* para los lenguajes StreamSQL y CCL. En contraposición, los operandos de condición —*EventProperty* y *Value*— sí que son semejantes en todos los lenguajes.

Con respecto a los operadores de patrón —*Every*, *EveryDistinct*, *FollowedBy*, *Range*, *Repeat*, *Until* y *While*— existen más dificultades a la hora de relacionarlos con sus análogos. El operador *Every*, que selecciona cada evento del tipo especificado, está presente en todos los lenguajes excepto en StreamSQL. Téngase en cuenta que aunque este último lenguaje sí que ofrece el operador *Every*, solo puede ser asociado con un intervalo de tiempo y, por tanto, no proporciona la misma funcionalidad del resto —en StreamSQL se selecciona por defecto cada evento, no siendo necesario el uso de un operador específico para tal fin.

Por otra parte, el operador *EveryDistinct* solo está disponible para los lenguajes EPL de Esper y CCL; sin embargo, podría definirse un comportamiento similar a dicho operador utilizando los operadores *Every*, *And* y *Not* como, por ejemplo, `EVERY a = Evento AND NOT b = Evento(b.id = a.id)`.

Puesto que los lenguajes EPL de Oracle, StreamSQL y CCL no hacen distinción entre los operadores *Range* y *Repeat*, se propone como solución el uso del operador `BETWEEN...AND...` para obtener el mismo comportamiento; salvo en el caso de CCL que sí ofrece `n ROWS` como operador de repetición.

De igual forma, tampoco existen equivalencias exactas para *Until* y *While*. Por un lado, el operador `UNTIL` de CCL solo puede indicar como condición de parada una marca temporal, mientras que el operador `Until` de Esper es más genérico, permitiendo establecer otro tipo de condiciones. Por otro lado, tampoco existe ningún operador idéntico al `While` de EPL de Esper, proponiéndose en su lugar operadores similares, aunque no idénticos, como son el `FOREACH` para StreamSQL y el `FOR` para CCL.

El resto de las metaclases presente en esta comparativa —operadores lógicos, de comparación, aritméticos y agregación—, así como todas las metaclases de ventanas de datos, disponen de equivalentes para todos los EPL analizados. Es más, algunas metaclases tienen más de un operador equivalente para un mismo lenguaje como ocurre, por ejemplo, con el operador *And*.

A partir de este análisis, puede concluirse que los elementos gráficos de los modelos que se definan haciendo uso del DSML de patrones de eventos podrán transformarse a distintos EPL, y no solamente al caso concreto de EPL de Esper. Para ello simplemente será necesario crear un nuevo módulo con las nuevas reglas de transformación para el EPL concreto que se desee utilizar, así como hacer uso de la API que proporcione el nuevo motor CEP para registrar los tipos de eventos simples y los patrones generados.

Por consiguiente, el DSML propuesto permitirá que los patrones de eventos sean defi-

nidos una única vez por los expertos en el domino o usuarios finales y, a partir de entonces, puedan ser transformados al EPL proporcionado por el motor CEP que se requiera utilizar en cada ocasión. Evidentemente, esto se traduce en un ahorro de tiempo muy significativo y, sobre todo, en la minimización del número de errores producidos a la hora de implementar los patrones de eventos, al ser un proceso totalmente automático, evitándose así la aparición de los fallos producidos por los propios programadores cuando escriben el código manualmente.

8.2. Editor Gráfico Reconfigurable para el Modelado y Generación de Código de Patrones de Eventos

En esta sección se evalúa el editor gráfico reconfigurable para el modelado y la generación de código de patrones de eventos desarrollo en esta tesis doctoral y, a continuación, se compara con otros editores existentes.

8.2.1. Funcionalidad y Usabilidad

La evaluación tanto del editor gráfico para el modelado de dominios CEP como del editor gráfico reconfigurable para el modelado y la generación de código de patrones de eventos se ha llevado a cabo a través de un cuestionario que se ha realizado a los usuarios finales. Puesto que el editor de patrones de eventos también incluye todas las funcionalidades ofrecidas por el editor de dominios CEP, se ha optado por realizar directamente la evaluación sobre el editor de patrones, para evitar sobrecargar a los encuestados.

Esta encuesta ha sido diseñada con el propósito de evaluar tanto la usabilidad de los editores —medición de la calidad de la interfaz de usuario— como su funcionalidad —el editor hace lo que se desea que haga. Esta encuesta está basada en la propuesta en [Sen12], con la que también persiguen evaluar un editor de patrones de eventos, denominado PANTEON, desarrollado dentro del proyecto europeo ALERT del FP7; así pues la utilidad del cuestionario se encuentra avalada por dichos estudios. Es importante destacar que este editor proporciona ventajas muy significativas sobre PANTEON como se demuestra en la Sección 8.2.2.

Se ha confeccionado un cuestionario formado por un total de 19 preguntas, disponible al completo en el Anexo A, para evaluar tanto la funcionalidad como la usabilidad del editor:

- *P1*: ¿Se considera un experto en el procesamiento de eventos complejos (CEP)?
- *P2*: ¿Es experto en el dominio para el que se van a definir los patrones de eventos?
- *P3*: ¿Cómo le gustaría definir los patrones de eventos?
- *P4*: ¿Qué tipo de usuario es el más adecuado para utilizar el editor de patrones de eventos que está evaluando?

- *P5*: ¿Está claro cuál es el propósito del editor?
- *P6*: ¿El editor ha satisfecho sus expectativas en general?
- *P7*: ¿Cuáles son las funcionalidades que el editor proporciona en cuanto a los dominios CEP?
- *P8*: ¿Cuáles son las funcionalidades que el editor proporciona en cuanto a los patrones de eventos?
- *P9*: ¿Considera útiles las funcionalidades proporcionadas por el editor?
- *P10*: ¿Considera irrelevantes algunas funcionalidades del editor?
- *P11*: ¿Considera irrelevantes algunas herramientas de la paleta gráfica del editor?
- *P12*: ¿Ha podido crear correctamente con el editor los patrones de eventos que se le han pedido para un dominio concreto?
- *P13*: Una vez definidos los patrones de eventos gráficamente, ¿tiene claro cuál es el propósito de dichos patrones?
- *P14*: ¿Qué nivel de destreza en lenguajes de procesamiento de eventos considera que se requiere para definir los patrones de eventos gráficamente?
- *P15*: ¿Cree que el editor podría reducir el tiempo que necesita para definir los patrones de eventos; en lugar de escribirlos "a mano", codificándolos mediante un lenguaje específico de procesamiento de eventos?
- *P16*: ¿Le ha sido fácil encontrar cada opción en el editor para crear los patrones de eventos gráficamente?
- *P17*: ¿La tarea que se le ha propuesto realizar con el editor se ha comprendido fácilmente?
- *P18*: ¿Cuánto tiempo (en minutos) ha empleado para llevar a cabo la tarea propuesta?
- *P19*: ¿Tiene alguna sugerencia para mejorar el editor?

Concretamente, esta evaluación ha sido realizada por 15 personas a las que se les ha solicitado definir gráficamente los tres patrones de eventos descritos en el caso de estudio de seguridad en redes (véase la Sección 7.2): *ICMP Echo Request*, *ICMP Echo Reply* y *ICMP Ping Response Time*. Para ello, se les ha proporcionado el dominio CEP de seguridad en redes, detectado automáticamente usando la opción *CEP Domain* > *Auto-detect Domain*, que han tenido que importar antes de comenzar con el modelado de los patrones para reconfigurar el editor de patrones; además, se les ha proporcionado una descripción textual de cada uno de estos patrones. Los encuestados han sido clasificados en 2 grupos:

- *Grupo 1 - Expertos en el dominio de seguridad pero no expertos en CEP*: este grupo se compone de un total de 13 estudiantes de la asignatura Seguridad y Competencias Profesionales (SCP) de la Titulación de Ingeniería en Informática de la UCA. Antes de realizar la evaluación, estos estudiantes han sido formados previamente en el ámbito de la seguridad durante 3 meses, por lo que son considerados expertos en seguridad. Sin embargo, no tienen ningún conocimiento sobre ningún EPL en particular.
- *Grupo 2 - Expertos en el dominio de seguridad y también expertos en CEP*: este grupo está formado por 2 investigadores, uno perteneciente a la *University of Applied Sciences Frankfurt am Main* (Alemania) y otro a la UCA, ambos expertos tanto en seguridad como en CEP.

Los resultados obtenidos de las encuestas realizadas se presentan en la Tabla 8.2. Todos los resultados se presentan en porcentajes (%), especificándose qué porcentaje representa cada respuesta de la pregunta en cuestión dentro del Grupo 1 y dentro del Grupo 2, así como el porcentaje de los que han seleccionado esa respuesta sobre el total de los 15 encuestados. A continuación, se detallan pormenorizadamente estos resultados.

Tabla 8.2: Resultados obtenidos de las encuestas realizadas.

Pregunta	**Respuestas**	**Grupo 1 (%)**	**Grupo 2 (%)**	**Total (%)**
P1	No	100	0	86,7
	Sí	0	100	13,3
P2	Sí	100	100	100
P3	Utilizando un editor gráfico	61,5	50	60
	Codificando el patrón de eventos usando un lenguaje de programación	7,7	0	6,7
	Utilizando un editor gráfico y también codificándolo usando un lenguaje de programación	30,8	50	33,3
P4	Usuarios del dominio de aplicación (sin necesidad de conocimientos de programación)	92,3	100	93,3
	Programadores	7,7	0	6,7
P5	Moderadamente	46,2	0	40
	Mucho	38,5	0	33,3
	Completamente	15,4	100	26,7
P6	Moderadamente	23,1	0	20

(Continúa en la página siguiente)

(Continúa de la página anterior)

Pregunta	**Respuestas**	**Grupo 1 (%)**	**Grupo 2 (%)**	**Total (%)**
	Mucho	69,2	100	73,3
	Completamente	7,7	0	6,7
P7	Todas las funcionalidades	100	100	100
P8	Todas las funcionalidades	100	100	100
P9	Sí	100	100	100
P10	No	100	100	100
P11	No	100	100	100
P12	Moderadamente	7,7	0	6,7
	Mucho	23,1	0	20
	Completamente	69,2	100	73,3
P13	Moderadamente	30,8	0	26,7
	Mucho	61,5	0	53,3
	Completamente	7,7	100	20
P14	Novato	15,4	50	20
	Principiante	61,5	50	60
	Competente	23,1	0	20
P15	Moderadamente	15,4	0	13,3
	Mucho	53,8	50	53,4
	Completamente	30,8	50	33,3
P16	Un poco	15,4	0	13,3
	Moderadamente	30,8	0	26,7
	Mucho	46,2	100	53,3
	Completamente	7,7	0	6,7
P17	Moderadamente	7,7	0	6,7
	Mucho	76,9	50	73,3
	Completamente	15,4	50	20
P18	10-20 min	7,7	0	6,7
	20-30 min	30,8	100	40
	30-40 min	53,8	0	46,6
	40-50 min	7,7	0	6,7
P19	No	69,2	0	60
	Sí	30,8	100	40

En primer lugar, puede comprobarse que los 13 estudiantes, no expertos en CEP, representan el 86,7 % mientras que los 2 investigadores son el 13,3 % del total. Eso sí, todos los encuestados son expertos en el dominio de seguridad.

Uno de los resultados a destacar es que el 60 % de los encuestados preferirían definir los patrones de eventos utilizando únicamente un editor gráfico, un 33,3 % preferirían utilizando tanto el editor gráfico como un lenguaje de programación —aunque dan prioridad al uso del editor gráfico sobre programarlos a mano—; tan solo un 6,7 % prefiere directamente

programar los patrones sin el uso de un editor gráfico. Estos resultados ponen de relieve la necesidad real de disponer de un editor para llevar a cabo la definición de estos patrones de una forma gráfica e intuitiva; avalándose así uno de los objetivos de esta tesis doctoral. Hay que tener en cuenta, además, que los encuestados son programadores. Se entiende que los expertos en el dominio no tienen por qué ser programadores y es esperable que, en ese contexto, el porcentaje que se decante por la herramienta gráfica sea mayor.

Además, todos los encuestados excepto uno (93,3 %) consideran que el tipo de usuario más adecuado para utilizar este editor de patrones son los usuarios del dominio de aplicación, es decir, aquellos que no tienen por qué disponer de conocimientos en programación. De hecho, este ha sido uno de los propósitos del editor: el desarrollo de un editor gráfico que abstraiga los detalles de implementación para que pueda ser usado por cualquier usuario no experto en las tecnologías requeridas.

Sin embargo, mientras que el Grupo 2 conoce claramente cuál es el propósito del editor (100 %), ya que son expertos en CEP, no ocurre lo mismo con el Grupo 1 donde el 46,2 % reconoce moderadamente el propósito del editor, el 38,5 % mucho y el 15,4 % completamente. Esta situación puede deberse a que, para evaluar tanto la usabilidad como la funcionalidad del editor, no se ha proporcionado a los encuestados un tutorial previo a la realización de la evaluación, con la intención de comprobar si realmente todas las opciones de los menús y las herramientas ofrecidas por el editor se presentan de una forma amigable e intuitiva. Además, estos usuarios no han tenido la necesidad, en su ámbito de trabajo, de hacer uso del procesamiento de eventos complejos, por lo que difícilmente pueden entender el propósito general del editor. No obstante, los resultados obtenidos son considerados muy positivos, y podría afirmarse que serían mejorables en el caso de que se proporcionase un manual de usuario del editor a los que no sean expertos en CEP —aunque no se haya considerado oportuno hacerlo de este modo para valorar cuál es realmente la experiencia del usuario con el editor.

Cabe destacar que el editor ha satisfecho las expectativas en general de la mayoría de los encuestados tanto del Grupo 1 como del Grupo 2, y la totalidad de los encuestados ha sido capaz de reconocer todas las funcionalidades proporcionadas tanto por el editor de dominios como por el editor de patrones, además de considerarlas útiles. Por otro lado, ninguno de los encuestados considera irrelevante ninguna de las funcionalidades del editor así como tampoco ninguna de las herramientas de su paleta gráfica.

En cuanto a si ha podido crear correctamente con el editor los patrones de eventos que se le han pedido para un dominio concreto, el Grupo 2 lo ha conseguido completamente, al igual que una gran mayoría del Grupo 1 (69,2 %) y un 23,1 % evaluado como mucho. Esto demuestra el buen grado de usabilidad y funcionalidad del editor, puesto que aun no siendo expertos en CEP, la mayor parte de los usuarios ha podido crear correctamente todos los patrones propuestos. En esta ocasión, también se estima que estos resultados podrían incrementarse si se proporcionase un manual de usuario.

Una vez definidos los patrones de eventos gráficamente, un buen número de usuarios del Grupo 1 tiene bastante claro cuál es el propósito de dichos patrones, frente al 30,8 % que lo tiene moderadamente claro. El no haber recibido una formación previa sobre CEP, ni haber tenido necesidad de usarlo previamente, puede ser la causa por la cual no comprenden a

la perfección la finalidad de estos patrones. Los usuarios del Grupo 2, debido a que sí son expertos en CEP, conocen de antemano la utilidad de estos patrones.

Por otra parte, el 60 % de todos los encuestados considera que el nivel de destreza en lenguajes de procesamiento de eventos requerido para definir los patrones de eventos gráficamente debe ser principiante, frente al 20 % que ha respondido novato y el otro 20 % competente. Esto pone de manifiesto la hipótesis que se ha formulado previamente: una formación muy básica en CEP en la que se definan conceptos básicos como patrones de eventos y eventos complejos, entre otros, sin mencionar detalles propios de implementación, mejoraría más la experiencia del usuario con el editor. Precisamente esta pueda ser la causa por la que dos usuarios del Grupo 1 (15,4 %) no hayan encontrado fácilmente cada opción en el editor para crear los patrones de eventos gráficamente, al no estar familiarizados, por ejemplo, con el uso de los operadores de patrones como pueden ser el *Every* o el *FollowedBy*, muy específicos del ámbito del procesamiento de eventos complejos.

En cuanto a la tarea que se les han propuesto por escrito para llevar a cabo la evaluación, la mayor parte de los encuestados la ha comprendido fácilmente.

Es esencial remarcar que el 86,7 % cree que el editor podría reducir muy considerablemente el tiempo que necesita para definir los patrones de eventos; en lugar de escribirlos "a mano", codificándolos mediante un lenguaje específico de procesamiento de eventos. Como puede comprobarse en la pregunta 18, la mayoría de los encuestados ha necesitado aproximadamente unos 35 minutos para definir los tres patrones de eventos, ya incluido también el tiempo inicial necesario para la familiarización con el editor —puede observarse que este tiempo se reduce en el caso de que se sea experto en CEP. Así pues, se trata de un dato muy significativo porque precisamente es una de las razones que han motivado el desarrollado de este editor gráfico de patrones. Desde la experiencia del propio doctorando y su dirección de proyectos fin de carrera a estudiantes de titulaciones de ingeniería informática, puede afirmarse rotundamente que el tiempo requerido para que cualquier persona, que no conozca CEP ni ningún lenguaje de procesamiento de eventos complejos, pueda llegar a definir la sintaxis correcta de un patrón de eventos es muy superior —al menos, del orden de días o semanas— al tiempo que se necesita haciendo uso del editor gráfico propuesto en esta tesis doctoral; más aún si el usuario final no es informático.

Pero esto no es todo, como se explicó en el enfoque propuesto en el Capítulo 3, la solución propuesta en esta tesis no solo permite obtener el código de los patrones de eventos sino también el código de las acciones a llevar a cabo en una SOA 2.0 cuando estos se detecten, así como su despliegue en tiempo de ejecución tanto en el motor CEP como en el ESB, respectivamente. El no hacer uso de esta solución implicaría que el usuario final también tendría que conocer el lenguaje XML a utilizar para definir dichas acciones manualmente, así como la gestión de dicha arquitectura y su integración con el motor CEP, lo que conllevaría un mayor incremento del tiempo estimado para definir los patrones de eventos manualmente; indudablemente, quedando fuera del alcance de un usuario que es experto en el dominio pero no en las tecnologías.

Finalmente, los encuestados realizaron algunas sugerencias que han sido introducidas en la última versión del editor, entre las que cabe destacar las siguientes para mejorar la usabilidad: 1) en el caso de que al validar un modelo de patrón de eventos existan errores,

además de que sean indicados gráficamente en los elementos del modelo, se abrirá automáticamente la ventana con el listado de errores; 2) al hacer doble clic sobre un elemento gráfico del modelo, se abrirá la ventana de propiedades para ese elemento; y 3) no eliminar ninguno de los menús proporcionados por el IDE Eclipse, sino simplemente añadir los nuevos menús personalizados del editor de patrones; de esta forma, aquellos usuarios que ya estén familiarizados con Eclipse podrán seguir trabajando con las opciones que ya hayan utilizado con anterioridad.

8.2.2. Un Estudio Comparativo con otros Editores Existentes

Actualmente existen algunos editores para la definición de patrones de eventos y su transformación a código EPL. No obstante, la gran mayoría de estos editores requiere a cualquier usuario no experto en CEP que escriba a mano, al menos, una parte del código EPL que implementa al patrón de eventos en cuestión.

Aparte de esta problemática, estos editores son desarrollados normalmente por grandes empresas como herramientas complementarias a sus sistemas de procesamiento de eventos complejos particulares como, por ejemplo, Oracle CEP Visualizer [Ora14], StreamBase Studio [Str14] y SAP Sybase ESP Studio [Syb14]; por lo cual suelen realizar únicamente transformaciones de los patrones al código específico que sea capaz de entender el sistema en cuestión. Esta dependencia entre motor CEP y editor podría requerir que el usuario deba definir n veces el mismo patrón de eventos para cada uno de los n distintos tipos de sistemas CEP en los que se necesite detectar el patrón. Lo que agrava todavía más esta situación es el hecho de que el usuario —o más bien el programador informático, debido a que normalmente se le exigirá escribir parte del código a mano— tendrá que invertir mucho tiempo en aprender cada uno de los lenguajes EPL así como familiarizarse con el uso de cada uno de los editores correspondientes. Precisamente, el editor de patrones desarrollado en esta tesis persigue solventar todos estos problemas mencionados, minimizando la curva de aprendizaje.

Así pues, se realiza un estudio comparativo del editor desarrollado en esta tesis con otros editores existentes. Tras un exhaustivo proceso de revisión bibliográfica, solo se han encontrado dos editores que podrían ser considerados como posibles competidores, atendiendo a la funcionalidad y usabilidad proporcionadas: el editor PANTEON [Sen12], desarrollado dentro del proyecto europeo ALERT del FP7 y el *CEP Editor* [Soc13a], desarrollado dentro del proyecto SocEDA financiado por la ANR. En la Tabla 8.3 se muestran los resultados obtenidos de este estudio comparativo que son descritos, a continuación, de forma pormenorizada.

Tabla 8.3: Comparativa del editor propuesto en esta tesis doctoral con otros existentes.

Nº	**Criterio**	**Propuesto**	**PANTEON**	***CEP Editor***
1	Editor gráfico basado en GMF	x		

(Continúa en la página siguiente)

(Continúa de la página anterior)

Nº	**Criterio**	**Propuesto**	**PANTEON**	***CEP Editor***
2	Editor desarrollado como una aplicación Eclipse	x		
3	Integración con un motor CEP	x	x	x
4	Integración con un ESB	x	x	x
5	Editor reconfigurable de dominios CEP	x		
6	Personalización de la paleta del editor en tiempo de ejecución	x	x	
7	Definición gráfica de dominios CEP	x		
8	Autodetección de dominios CEP y representación gráfica automática	x		
9	Definición de eventos con propiedades anidadas	x		x
10	Validación de los modelos gráficos de dominio CEP	x		
11	Almacenamiento de dominios CEP	x	x	x
12	Personalización de dominios CEP con imágenes	x		
13	Importación y exportación de dominios CEP	x		x
14	Definición gráfica de patrones de eventos	x	x	x
15	Definición gráfica de acciones para los patrones	x	x	
16	Disponibilidad de un gran número de operadores, operandos y ventanas de datos	x		
17	Elementos representados como nodos gráficos	x	x	x
18	Definición de jerarquía de eventos complejos de forma intuitiva	x	x	
19	Validación de los modelos gráficos de patrones	x		
20	Almacenamiento de los patrones	x	x	x
21	Transformación de patrones de eventos a código	x	x	x
22	Personalización de patrones con imágenes	x		

(Continúa en la página siguiente)

(Continúa de la página anterior)

Nº	Criterio	Propuesto	PANTEON	*CEP Editor*
23	Importación y exportación de patrones	x		x
24	Despliegue de los patrones en el motor CEP en tiempo de ejecución	x	x	x
25	Despliegue de las acciones en el ESB en tiempo de ejecución	x	x	x
26	Notificación de eventos complejos mediante distintos servicios	x	x	x
27	Editor ofrece un buen grado de usabilidad	x		
28	Recomendación de patrones de eventos de forma semi-automática		x	

Editor PANTEON del Proyecto ALERT

El editor PANTEON no es una aplicación Eclipse ni está basado en GMF, sino en GWT, a diferencia del desarrollado en esta tesis. Precisamente una de las ventajas de desarrollar una aplicación Eclipse es la posibilidad de ser integrada con otras aplicaciones ya existentes, puesto que se trata de uno de los IDE más conocidos y utilizados a nivel mundial. Por otra parte, como ya se ha mencionado, GMF es un *plug-in* Eclipse que permite el desarrollo de editores gráficos de modelos a partir de metamodelos; estos metamodelos se especifican haciendo uso de EMF. Por consiguiente, cuando se diseña un patrón de eventos con el editor desarrollado en esta tesis se crean dos ficheros: un modelo EMF serializado en formato XMI y un diagrama, que almacena toda la información relativa a la parte visual del patrón como, por ejemplo, la posición, el color y el tamaño de los elementos gráficos. Gracias a la especificación XMI pueden compartirse los modelos entre diferentes herramientas de modelado. Sin embargo, al diseñar un patrón de eventos con PANTEON se genera un fichero XML Schema interno definido por los propios desarrolladores que, a diferencia de XMI, presenta más dificultades a la hora de compartir los patrones entre distintas herramientas.

PANTEON está integrado con ETALIS [Ani14], como se ha comentado en el Capítulo 2 es un sistema de código abierto para CEP implementado en Prolog y que ofrece dos lenguajes basados en reglas: ELE y EP-SPARQL. Concretamente, ETALIS utiliza semánticas declarativas basadas en programación lógica. Los eventos complejos son generados a partir de eventos simples por medio de reglas deductivas. Además, PANTEON está integrado con el ESB Petals [Pet14], un ESB de código abierto desarrollado por OW2 Consortium; mientras que el editor aquí propuesto utiliza el motor CEP Esper y el ESB Mule, ambos también de código abierto. Uno de los grandes beneficios de utilizar Mule es que proporciona su herramienta gráfica MuleStudio, basada en Eclipse, que genera automáticamente el código XML necesario para implementar los flujos Mule que el usuario previamente ha-

ya diseñado gráficamente, reduciendo la curva de aprendizaje y la aparición de errores al escribir manualmente el código.

El editor PANTEON no dispone de un editor para la definición gráfica de dominios CEP, como sí ofrece el editor propuesto. Los tipos de eventos se definen usando ontologías que son, posteriormente, recibidos por el editor a través de un servicio de metadatos. Este editor se encargará de transformar los tipos de eventos en un XML Schema interno para que pueda ser visualizado gráficamente. Sin embargo, no es posible la definición gráfica de dominios CEP, con los distintos tipos de eventos que los forman, desde el propio editor; tan solo pueden usarse los tipos de eventos que se encuentren disponibles en ese momento como herramientas en la paleta del editor. Además, no se permiten definir tipos de eventos que tengan propiedades anidadas. Por otra parte, la paleta del editor puede personalizarse con los tipos de eventos recibidos, aunque no parece que el editor pueda reconfigurarse tan solo con los tipos de eventos que pertenezcan a un dominio en particular.

Este editor sí que permite definir los patrones de eventos gráficamente, incluso las acciones. Sin embargo, presenta algunas limitaciones con respecto al editor propuesto en esta tesis que se enumeran a continuación.

Aunque los elementos del patrón son representados como nodos gráficos interconectados, estos nodos son realmente ventanas compuestas de botones y listas desplegables a través de las cuales el usuario final puede ir añadiendo las propiedades de los eventos que desea utilizar y establecer, así como las condiciones (véase la Figura 8.1, extraída de [Sen12]). Esto difiere de cómo se modela con el editor que se compara: cuando un usuario selecciona un tipo de evento de la paleta y lo arrastra hasta el canvas, automáticamente se visualizará el evento con todas sus propiedades para las cuales el usuario podrá directamente establecer condiciones enlazándolas con los operadores correspondientes.

Por otro lado, esta representación gráfica de los patrones impide que se puedan definir condiciones más complejas, como será necesario en patrones de mayor envergadura. En cuanto a la definición del tipo de evento complejo que se debe crear al detectar el patrón, se realiza de manera similar que para los eventos simples, esto es, utilizando una ventana de tipo *output* con las propiedades de eventos complejos —parecida a la ventana de evento simple *ALERT:New Issue - ID:1* disponible en la Figura 8.1. Esta definición de tipos de evento complejo es muy diferente a lo que sí se puede realizar con el editor propuesto: las propiedades de eventos complejos pueden ser calculadas a partir de distintas operaciones matemáticas, o bien enlazándolas directamente con las propiedades de otros eventos (véase, por ejemplo, la Figura 7.14). Además, solamente se encuentran disponibles 5 tipos de operadores —Filter, AND, OR, NOT, SEQ— en comparación con los operadores de patrón, lógicos, relacionales y de agregación del editor propuesto; ni tampoco proporciona ventanas temporales. Conviene señalar que la representación escogida en PANTEON puede dar lugar a gráficos de patrones muy grandes y poco amigables; obsérvese, por ejemplo, el tamaño de la ventana NOT.

Puesto que los tipos de eventos complejos definidos se incluyen en la paleta del editor, PANTEON permite la definición de jerarquía de eventos complejos. También valida y almacena los patrones; ahora bien, el editor no dispone de ninguna opción para validar el modelo gráfico diseñado, ya que la etapa de validación se lleva a cabo al transformar los

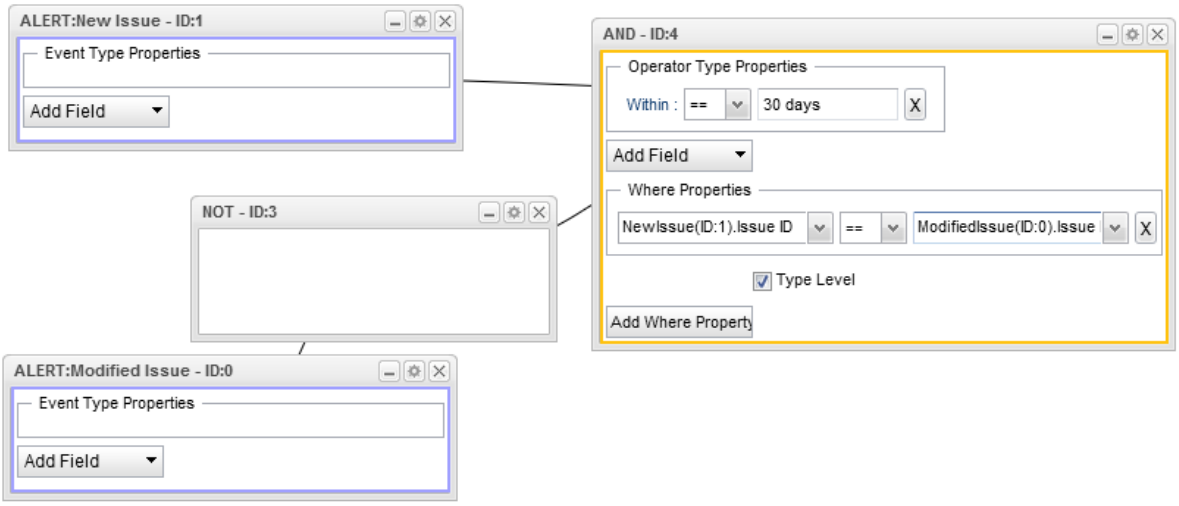

Figura 8.1: Definición gráfica de un patrón de eventos utilizando el editor PANTEON del proyecto ALERT.

patrones a código. Otras de las limitaciones del editor son que no permite la personalización de los elementos gráficos mediante el uso de imágenes, lo que mejoraría la experiencia del usuario, ni tampoco proporciona opciones de exportación e importación de los patrones.

En cuanto a aspectos destacables, el editor sí que permite el registro de los patrones y de las acciones en el sistema, concretamente en ETALIS y Petals, respectivamente, y es capaz de notificar los eventos complejos a través de servicios como el del correo electrónico.

Una de las características más relevantes que dispone el editor PANTEON, y que el editor desarrollado en esta tesis todavía no proporciona, es la recomendación de patrones de eventos de forma semi-automática, que permite al usuario final crear nuevos patrones reutilizando otros patrones de eventos, recomendados por el propio editor a partir del análisis y la interpretación de las estadísticas sobre la ejecución de estos patrones existentes.

Editor CEP del Proyecto SocEDA

El editor de patrones *CEP Editor* tampoco es una aplicación Eclipse ni está basado en GMF; se ha implementado con GWT, al igual que el editor PANTEON. Cuando se diseña un patrón de eventos con este editor se genera el código EPL que será registrado en una base de datos de patrones. En el caso de que se desee exportar el patrón, entonces se creará un fichero XML con la representación del diagrama y otro con la configuración realizada sobre los distintos componentes. Por lo cual, presenta más dificultades a la hora de compartir los patrones entre distintas herramientas, como no ocurriría si se exportase en formato XMI.

Este editor está integrado con el motor CEP Esper, por lo tanto, el código que genera automáticamente está en formato EPL de Esper. Aunque utiliza el mismo motor CEP que el editor de patrones desarrollado en esta tesis, no utiliza Mule como ESB, sino el mismo

ESB que utiliza el editor PANTEON: Petals.

Para gestionar los tipos de eventos, se utiliza un repositorio. Cada tipo de evento tiene un nombre y un identificador, y sus propiedades se registran como tuplas de tablas en las que se detallan el identificador, el nombre de la propiedad, así como el tipo de dato Java o el nombre de la clase definida por el propio programador. Al poder definir las propiedades como tipos de clases Java, sí se permite la descripción de propiedades anidadas.

Este editor de patrones tampoco dispone de un editor para la definición gráfica de dominios CEP. En la Figura 8.2, extraída de [Soc13b], se muestra una imagen de este editor. Como puede observarse, la paleta es muy simple donde sus elementos son clasificados en 4 categorías: operadores (*operators*), flujos (*streams*), ventanas (*windows*) y herramientas (*tools*). Para especificar un tipo de evento en el patrón a definir gráficamente, se usa la herramienta *input* que será posicionada sobre la zona de diagrama del editor. En las propiedades de este elemento podrá seleccionarse, mediante una lista desplegable de opciones, uno de los tipos de eventos que hayan sido recuperados tras la consulta a dicho repositorio de tipos de eventos. Así pues, queda patente que la paleta del editor es totalmente estática y no es modificable en tiempo de ejecución. Esto también dificulta la posibilidad de definir jerarquías de eventos complejos de una forma gráfica e intuitiva.

Al igual que el editor propuesto, este editor hace posible el diseño de patrones de eventos gráficamente; lo que no ofrece es la posibilidad de definir gráficamente las acciones asociadas a los patrones, debiendo especificarse a través de una serie de botones y listas desplegables en modo texto. A pesar de que los elementos del patrón son representados como nodos gráficos interconectados, estos nodos son demasiado genéricos —existen únicamente 6 tipos de herramientas en la paleta—, repercutiendo negativamente en la usabilidad del editor. Por ejemplo, cualquier condición que deba cumplirse en el patrón, a excepción de las que impliquen el uso de una operación de unión o agregación, requerirá que se escriba textualmente su código EPL de Esper dentro de la ventana de propiedades del nodo de tipo *query*; lo cual hace que en última instancia el usuario final sí que deba conocer la sintaxis del lenguaje tratándose, por tanto, de una de las limitaciones más destacables en cuanto a la usabilidad de este editor.

Por consiguiente, esta representación de los patrones impide que se puedan definir condiciones más complejas en modo gráfico, como será necesario en patrones de mayor envergadura. Además, la definición del tipo de evento complejo que debe crearse al detectar el patrón, se realiza de la misma forma que para los eventos simples, esto es, utilizando un nodo de tipo *output* —similar al nodo *input*— concretando las propiedades de eventos complejos mediante una ventana de cuadros de texto y listas desplegables; muy diferente a lo que sí se puede realizar con el editor propuesto donde este tipo de propiedades puede ser calculada a partir de distintas operaciones matemáticas, o bien conectándolas directamente con las propiedades de otros eventos. Es importante destacar que la representación escogida para este editor puede dar lugar a gráficos poco intuitivos, ya que la mayor parte de las condiciones se encontrarán definidas en las ventanas de las propiedades de los elementos gráficos diseñados.

Otra de las limitaciones presentes en este editor es que la validación del patrón de eventos no se lleva a cabo sobre el propio modelo diseñado, sino una vez haya sido transformado

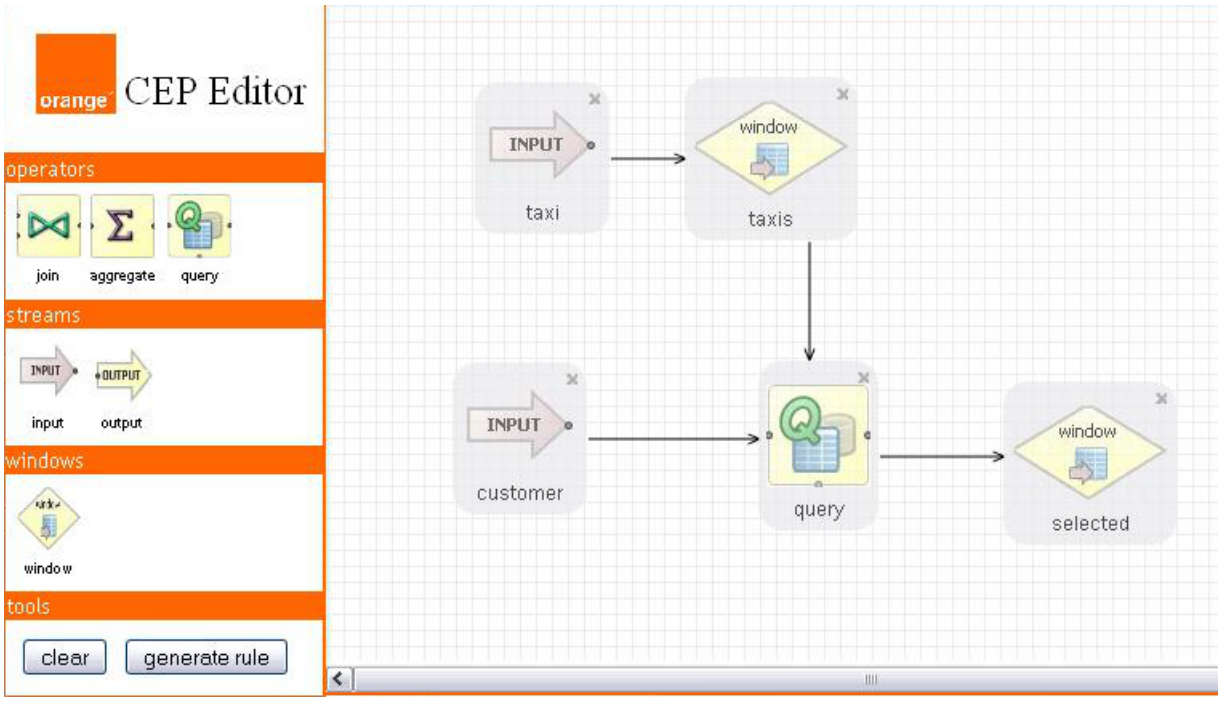

Figura 8.2: Definición gráfica de un patrón de eventos utilizando el *CEP Editor* del proyecto SocEDA.

a código EPL, momento en el cual un *servlet* Java conectado al motor CEP comprobará si Esper devuelve algún mensaje de error de sintaxis incorrecta. En caso afirmativo, se informará al usuario de los errores presentes en el patrón modelado. Evidentemente, esto conlleva una sobrecarga del motor y un retraso de tiempo bastante considerable durante el modelado de los patrones, como consecuencia de las continuas peticiones de validación que podrían efectuarse por parte del usuario, a través del editor, con el propósito de ir confirmando si el modelo que está diseñando es correcto o no.

Este editor tampoco posibilita la personalización de los elementos gráficos mediante el uso de imágenes, lo que mejoraría la experiencia del usuario.

En cuanto a aspectos positivos, el editor sí que permite el registro de los patrones y de las acciones en el sistema —en Esper y Petals, respectivamente—, es capaz de notificar los eventos complejos a través de servicios como el del correo electrónico, y permite la exportación e importación de los patrones de eventos.

8.3. Enfoque Dirigido por Modelos para CEP en SOA 2.0

En esta sección se evalúa el enfoque dirigido por modelos para la integración de CEP y SOA 2.0 propuesto en esta tesis doctoral, estudiando su viabilidad y el incremento de

la productividad. Para ello, se han analizado los dos casos de estudio sobre domótica y seguridad en redes descritos en el Capítulo 7, midiéndose las líneas de código o LOC (*Lines of Code*) —sin comentarios ni líneas en blanco— que han sido necesarias generar para cada caso de estudio. Por un lado, la Tabla 8.4 presenta el número de líneas del código generado tanto automáticamente como el implementado manualmente para el caso de estudio sobre domótica, mientras que la Tabla 8.5 recoge dicha medición para el caso de estudio de seguridad en redes.

Tabla 8.4: Número de líneas del código generado para el caso de estudio sobre domótica.

	Automáticamente					**Manualmente**
	XMI		**EPL**	**XML**	**Total**	**Java**
	Modelo	**Diagrama**				
Flujo de recepción y gestión de eventos	0	0	0	9	9	65
Dominio CEP	17	5	0	0	22	0
Patrón *Irresponsible-EnergyConsumption*	39	334	7	25	405	0
Patrón *Fire*	67	571	8	19	665	0
Patrón *PowerFailure*	64	554	8	22	648	0
Patrón *IrresponsibleTelevisionUse*	66	558	6	22	652	0
Eventos complejos	28	5	0	0	33	0

En ambas tablas, el código generado automáticamente se ha clasificado según el lenguaje de programación en el que se ha implementado. Así pues, el código XMI engloba las LOC tanto de los modelos EMF (columna 2 de las tablas) como de sus diagramas correspondientes (columna 3 de las tablas) generados automáticamente por el editor gráfico de dominios CEP cuando el usuario final efectúa el modelado del dominio CEP —o es autodetectado en tiempo de ejecución— y por el editor reconfigurable de patrones de eventos cuando el usuario define gráficamente estos patrones. Nótese que el modelado de patrones de eventos también genera internamente un modelo EMF, y su correspondiente diagrama, con el tipo de evento complejo definido en cada uno de estos patrones; esto es necesario para la personalización dinámica de la paleta del editor. En la columna 4 de las tablas se indica las LOC del código EPL generado automáticamente por cada patrón modelado —que será insertado posteriormente en el motor Esper— y en la columna 5 se muestra el código XML generado que implementa las acciones especificadas para cada uno de dichos patrones —que será ejecutado seguidamente en el ESB Mule. Finalmente, en la columna 6 se presenta el total de LOC generado automáticamente frente al total de LOC implementado manualmente en Java.

Como puede observarse, el único código que ha sido necesario implementar manualmen-

Tabla 8.5: Número de líneas del código generado para el caso de estudio sobre seguridad.

	Automáticamente					**Manualmente**
	XMI		**EPL**	**XML**	**Total**	**Java**
	Modelo	**Diagrama**				
Flujo de recepción y gestión de eventos	0	0	0	5	5	0
Dominio CEP	31	356	0	0	387	0
Patrón *icmp_echo_request*	45	472	7	22	546	0
Patrón *icmp_echo_reply*	45	472	7	22	546	0
Patrón *icmp_ping_response_time*	46	385	8	25	464	0
Eventos complejos	20	83	0	0	103	0

te es el correspondiente a la creación del flujo Mule de la SOA 2.0 propuesta que se encarga de la recepción y gestión de los eventos en cada caso de estudio, puesto que este flujo es dependiente del productor de eventos que se utilice en cada caso, tal y como se mencionó en la Sección 6.4. El transformador en Java que se ha creado en el primer caso de estudio para transformar la información proveniente de Xively a eventos *maps* de Java está compuesto por un total de 65 LOC. Realmente se trata del único código implementado manualmente, puesto que las 9 líneas XML que definen este flujo ESB así como las 5 líneas XML que definen el flujo ESB del segundo caso de estudio han sido generadas automáticamente por la herramienta Mule Studio al definir estos flujos gráficamente.

En resumen, para el primer caso de estudio se ha generado automáticamente un total de 2434 LOC y se han implementado manualmente 65 LOC, esto es, un 97,4 % de LOC generadas automáticamente frente a un 2,6 % implementadas manualmente; mientras que para el segundo caso de estudio se ha generado automáticamente el 100 % de las LOC.

8.4. Conclusiones

En este capítulo se ha realizado una evaluación de todas las aportaciones realizadas en esta tesis doctoral: un enfoque dirigido por modelos definido e implementado para el procesamiento de eventos complejos en SOA 2.0, un DSML y un editor gráfico para la definición de dominios CEP, y un DSML y un editor gráfico para la definición de patrones de eventos y su generación a código.

A partir de todos los resultados obtenidos, se han llegado a las siguientes conclusiones:

- Los DSML descritos para modelar los dominios CEP así como los patrones de eventos son lo suficientemente independientes de cualquier EPL, de forma que los modelos conformes a sus metamodelos puedan ser transformados a distintos EPL, y no solo

al lenguaje EPL de Esper.

- El código de las condiciones del patrón en EPL y de las acciones en XML generado automáticamente por el editor de patrones es totalmente ejecutable en el motor Esper y desplegable en el ESB Mule, respectivamente, es decir, el editor es capaz de generar el 100 % del código EPL y XML necesario. Es más, se puede afirmar que este código está libre de errores, debido al proceso de validación de los modelos gráficos efectuado previamente a la transformación de los modelos a código. Evidentemente, tanto el proceso de validación como el de transformación ha sido probado con distintos casos de prueba en los que se han ido comprobando si los resultados obtenidos eran iguales a los esperados. Gracias a esta generación automática de código, se evita al usuario final la curva de aprendizaje en CEP y en SOA 2.0 por la que debería pasar hasta ser capaz de implementar y desplegar un patrón libre de errores junto con sus acciones asociadas.

- El buen grado de funcionalidad y de usabilidad de los editores gráficos de modelado hace posible que no se requiera llevar a cabo ninguna tarea adicional manualmente como, por ejemplo, la modificación del contenido o extensión de ficheros o de la ubicación a otros directorios.

- El enfoque dirigido por modelos para CEP en SOA 2.0 es escalable y prácticamente independiente del dominio al que se aplique, a excepción del flujo que recibe la información del productor de eventos que podría requerir la implementación manual, por ejemplo de algún transformador, como ha ocurrido en el caso de estudio de domótica. Aún así, esto es insignificante comparado con todo el código generado automáticamente. En otras ocasiones, no tendrá por qué implementarse ningún código a mano, como queda demostrado con el caso de estudio de seguridad en redes.

Capítulo 9

Conclusiones y Trabajo Futuro

«La verdadera investigación consiste en buscar a oscuras el interruptor de la luz. Cuando la luz se enciende, todo el mundo lo ve muy claro.»
(Anónimo)

En este último capítulo se describen los objetivos logrados, las contribuciones y las conclusiones extraídas tras la realización de esta tesis doctoral. Asimismo, se exponen las líneas de trabajo futuras que se derivan de la presente investigación.

9.1. Contribuciones y Conclusiones

En el primer capítulo de esta disertación se establecieron los objetivos de esta tesis doctoral encaminados al desarrollo dirigido por modelos de interfaces específicas de dominio para CEP en SOA 2.0, con el objeto de facilitar a los expertos en el negocio tanto la definición de los patrones que necesiten detectar en sus sistemas de información, como la definición de las alertas que deban notificarse en tiempo real. Todo ello de una forma gráfica, amigable e intuitiva, y haciendo los detalles de implementación totalmente transparentes para estos usuarios.

A continuación, se detallan las principales contribuciones de esta tesis doctoral, así como las conclusiones extraídas tras la consecución de cada uno de los objetivos establecidos.

Estado del Arte sobre Enfoques Existentes para CEP, Editores Gráficos y Propuestas para la Integración de CEP con EDA y/o SOA

El primer objetivo que se planteó en esta tesis fue la recopilación del estado del arte de los enfoques existentes para CEP, los editores gráficos para la definición y la generación de código de patrones de eventos, así como las propuestas para la integración de CEP con EDA y/o SOA. Este objetivo se ha llevado a cabo mediante la realización de una extensa revisión bibliográfica (véase el Capítulo 2).

En cuanto a estos enfoques para CEP, se han encontrado trabajos que utilizan ontologías para la representación de eventos y patrones de eventos y, además, trabajos que, al igual que en esta tesis doctoral, proponen enfoques dirigidos por modelos.

Todos los enfoques analizados presentan limitaciones con respecto al propuesto en esta tesis. Por un lado, los enfoques ontológicos son incompletos en cuanto a los tipos de operadores y ventanas temporales que proporcionan, lo cual implica una limitación en la expresividad de los patrones más complejos que se requiera modelar. Además, la forma en que se han estructurado algunos de estos modelos ontológicos dificulta la escalabilidad y la comprensión de los patrones definidos.

Por otro lado, los trabajos sobre los enfoques dirigidos por modelos se han clasificado en dos categorías según la notación usada para representar los patrones de eventos: textual o gráfica. Ninguno de ellos ofrece la posibilidad de describir un dominio CEP compuesto de un conjunto de tipos de eventos, junto con su descripción y fecha de creación, lo cual facilitaría que los dominios pudiesen compartirse entre distintas herramientas de modelado que, incluso, podrían pertenecer a distintos usuarios.

Los enfoques que utilizan una representación textual de los patrones presentan un grado de usabilidad y funcionalidad inferior al propuesto en esta tesis. Algunas de las limitaciones más destacables son las siguientes: 1) se requiere crear un nuevo DSML por cada dominio donde se necesite aplicar CEP, lo que implica una mayor carga de trabajo y esfuerzo adicional por cada nuevo lenguaje a implementar; 2) se proporciona un conjunto predefinido de tipos de eventos para un dominio en particular —sin opción a modificarse para otros dominios—; 3) y no se da soporte a las acciones que han de ejecutarse cuando se detecten las situaciones de interés.

Asimismo, los enfoques que usan una representación gráfica de los patrones de eventos también presentan limitaciones, en cuanto a funcionalidad y usabilidad, en comparación con el propuesto en esta tesis doctoral. Por ejemplo, mientras que en los enfoques analizados se ha optado por el uso de notaciones en formato tabla con textos para definir los patrones de eventos, en el enfoque de la tesis se ha hecho uso de nodos y enlaces gráficos, un formato más gráfico e intuitivo para cualquier usuario, independientemente de su destreza ante la tecnología CEP.

En cuanto a los editores gráficos existentes para la definición y generación de código de patrones de eventos, estos no proporcionan todas las funcionalidades ni el buen grado de usabilidad de los editores construidos en esta tesis, tal y como se expone en el Capítulo 8.

Finalmente, tampoco se han encontrado propuestas para la integración de CEP con SOA 2.0 que proporcionen los beneficios de la solución tecnológica propuesta en el Capítulo 6. Estas propuestas en su mayoría son dependientes del dominio de aplicación y no ofrecen editores gráficos para facilitar al usuario la definición de dominios CEP y patrones de eventos.

Enfoque Dirigido por Modelos para el Procesamiento de Eventos Complejos en SOA 2.0

El segundo objetivo que se estableció en esta tesis doctoral fue definir un enfoque dirigido por modelos que posibilitase el procesamiento de eventos complejos en SOA 2.0, minimizando la curva de aprendizaje en las tecnologías requeridas para la detección de situaciones críticas o relevantes en tiempo real por parte del usuario final.

Este objetivo se ha satisfecho plenamente a través de la principal aportación de esta tesis doctoral, esto es, un enfoque dirigido por modelos para CEP en SOA 2.0 que es independiente tanto del dominio donde deba aplicarse CEP, como de los lenguajes requeridos tanto para implementar los patrones de eventos como las acciones encargadas de notificar las situaciones detectadas (véase el Capítulo 3).

Este enfoque está compuesto de dos partes bien diferenciadas. La primera parte hace referencia a todo el proceso que se lleva a cabo en tiempo de diseño mientras que la segunda parte engloba el proceso que se desarrolla en tiempo de ejecución.

Concretamente, en tiempo de diseño podrá modelarse un dominio CEP, creándose automáticamente a partir de la inferencia de los eventos que fluyan por una SOA 2.0 con la que el editor gráfico de dominios CEP esté conectado, o bien manualmente por parte del experto en el negocio haciendo uso de dicho editor. A partir del dominio CEP definido, el usuario final podrá modelar los patrones de eventos correspondientes a las situaciones de interés que desee detectar, así como las acciones a llevar a cabo. Posteriormente, estos modelos se validarán y transformarán automáticamente a código, que se desplegará en tiempo de ejecución tanto en un motor CEP como en un ESB; todo esto de forma transparente al usuario final y gracias al uso de técnicas de MDD.

DSML y Editor Gráfico para la Definición de Dominios CEP

El tercer objetivo definido se centraba en la definición de un DSML y la construcción de un editor gráfico de dominios CEP que facilitasen a cualquier usuario, experto en un dominio concreto pero no en CEP, la descripción de los tipos de eventos y propiedades particulares de dicho dominio.

Este objetivo se ha materializado en la definición de un metamodelo de dominios CEP, junto con sus restricciones, y la creación de un editor gráfico que da soporte al modelado de estos dominios (véase el Capítulo 4). El método de desarrollo que se ha seguido para construir el editor es el siguiente: 1) creación del metamodelo haciendo uso de Emfatic y Ecore; 2) generación de una versión inicial del editor utilizando GMF y EuGENia; 3) personalización del editor, modificando su paleta de herramientas, así como añadiendo un menú de opciones para que el usuario final pueda llevar a cabo el modelado de dominios CEP de forma intuitiva; 4) implementación de las reglas de validación, haciendo uso del lenguaje EVL, para comprobar si un dominio CEP modelado está bien formado; y 5) creación de una aplicación Eclipse RCP con el fin de que el editor pueda ejecutarse fuera del IDE Eclipse y en múltiples plataformas.

Una de las aportaciones más importantes que ofrece este DSML es la unificación de

la descripción de los dominios CEP —tipos de eventos y propiedades— mediante el uso de modelos, abstrayendo a los expertos en el dominio de los detalles de implementación necesarios para definir dichos dominios. Por otro lado, el editor construido permite el diseño gráfico y la validación automática de estos dominios CEP, así como la exportación e importación de estos para que puedan ser compartidos y reutilizados por otros expertos en el dominio.

DSML y Editor Gráfico Reconfigurable para la Definición y la Generación de Código de Patrones de Eventos

El cuarto objetivo definido consistía en la definición de un DSML y la construcción de un editor gráfico de patrones de eventos que permitiesen expresar mediante modelos gráficos e intuitivos qué situaciones se desean detectar para un dominio en particular y si estas deben ser notificadas a los usuarios interesados mediante algún servicio como, por ejemplo, de correo electrónico o de redes sociales.

Este objetivo se ha superado satisfactoriamente con la definición de un metamodelo de patrones de eventos, junto con sus restricciones, y la construcción del editor gráfico que puede reconfigurarse a partir de un dominio CEP, modelado por un experto en el dominio (véase el Capítulo 5). El método de desarrollo que se ha seguido para construir el editor es el siguiente: 1) creación del metamodelo haciendo uso de Emfatic y Ecore; 2) generación de una versión inicial del editor utilizando GMF y EuGENia; 3) personalización del editor, modificando su paleta de herramientas y añadiendo un menú de opciones para que el usuario final pueda llevar a cabo el modelado de patrones de eventos de forma intuitiva, así como introduciendo modificaciones en Java para que el editor pueda reconfigurarse automáticamente a partir de distintos modelos de dominio CEP; 4) implementación de las reglas de validación, haciendo uso del lenguaje EVL, para comprobar si un un patrón de eventos modelado está bien formado; 5) implementación de las reglas de transformación de modelo a código, mediante el lenguaje EGL, que permitirán transformar un modelo de patrón de eventos al código que lo implementa; y 6) creación de una aplicación Eclipse RCP con el fin de que el editor pueda ejecutarse fuera del IDE Eclipse y en múltiples plataformas.

Uno de los aspectos más relevantes de este DSML es que facilita el diseño de patrones totalmente independientes de su implementación. Gracias al MDD utilizado en el enfoque propuesto, cada patrón de eventos ha de definirse gráficamente una única vez, y podrían facilitarse transformaciones a posteriori tanto a distintos EPL —EPL de Esper, StreamSQL o CCL, entre otros— como al código que permita ejecutar sus acciones asociadas, por ejemplo, XML. Para ello, se ha creado el editor gráfico de patrones que hace posible que los usuarios noveles en CEP puedan concentrarse en el modelado de las situaciones a detectar, así como de las acciones a llevar a cabo, abstrayéndoles de todos los detalles de implementación. De esta forma, los modelos de patrones obtenidos y validados podrán exportarse e importarse, así como reutilizarse en distintos sistemas de información, mediante el proceso de transformación de estos modelos tanto al código EPL exigido por el motor CEP como al código requerido por el ESB con el que esté conectado el motor. En este caso, se ha

facilitado las reglas de transformación para el EPL de Esper y las reglas de transformación para XML.

Conviene destacar que el hecho de que el editor pueda reconfigurarse a partir de un modelo de dominio CEP, modificándose dinámicamente la paleta de herramientas dependiendo del dominio en cuestión, hará posible que el usuario disponga en todo momento de una interfaz gráfica adaptada al contexto específico para el que desee definir los patrones de eventos. Así pues, el editor pone al alcance de cualquier usuario no tecnólogo la definición de las situaciones críticas o relevantes que necesite detectar en tiempo real.

Solución Tecnológica para la Integración del Procesamiento de Eventos Complejos con una SOA 2.0

El quinto objetivo perseguía la integración de CEP con SOA 2.0, así como con los editores gráficos de modelado para hacer realidad el enfoque dirigido por modelos propuesto.

Este objetivo se ha logrado mediante el diseño y la implementación de una solución tecnológica para la integración de CEP con SOA 2.0, cuyo elemento clave es un ESB que actúa como capa de integración, permitiendo la combinación de CEP con SOA y EDA. En este contexto, se han descrito los principales componentes integrables en la SOA 2.0, clasificados según sean productores de eventos —generadores de eventos, aplicaciones, servicios web, sensores, plataformas IoT y servicios de redes sociales— o consumidores de eventos —consolas de monitorización, aplicaciones, servicios web, actuadores, plataformas IoT, servicios de redes sociales y de correo electrónico, y bases de datos.

Además, se han establecido las funcionalidades implicadas en la integración de CEP con SOA 2.0: recepción de eventos, transformación de eventos, filtrado de eventos, generación dinámica del dominio, envío de eventos al motor CEP, adición de patrones de eventos en el motor CEP, adición de acciones para patrones de eventos en el ESB, detección de patrones de eventos, recepción de eventos complejos, así como toma de decisiones (acciones para eventos complejos detectados).

Una vez diseñada dicha solución tecnológica, se ha implementado e integrado con los editores gráficos de modelado haciendo uso del ESB Mule y el motor CEP Esper.

En comparación con otras arquitecturas existentes, algunas de las funcionalidades más destacables de esta solución tecnológica son las siguientes: la integración de Mule con Esper, la generación automática del domino y la integración con los editores gráficos de modelado de dominio CEP y de patrones de eventos desarrollados en esta tesis doctoral, así como la inserción automática y en tiempo de ejecución tanto del código EPL de los patrones de eventos en el motor CEP como del código XML de las acciones a ejecutar en el ESB cuando se detecten los patrones; todo ello sin que se requiera ninguna modificación adicional por parte del usuario en la implementación de la arquitectura propuesta.

Asimismo, se han propuesto dos casos de estudio, uno en el ámbito de la domótica y otro en el ámbito de la seguridad en redes, a partir de los cuales se ha demostrado que dicha solución puede aplicarse a distintos dominios de aplicación, obteniéndose un conjunto de patrones de eventos para la detección de situaciones relevantes en ambos campos de aplicación.

Conclusiones de la Evaluación

Finalmente, el sexto objetivo consistía en la evaluación tanto del enfoque dirigido por modelos y los DSML propuestos, como de la funcionalidad y usabilidad de los editores gráficos.

Este objetivo se ha cumplido a través del estudio y la discusión de la viabilidad del enfoque dirigido por modelos propuesto y de los DSML, y mediante la evaluación de la satisfacción de los usuarios en cuanto a la funcionalidad y usabilidad de los editores gráficos.

Los resultados de la evaluación permiten establecer las siguientes conclusiones parciales:

- Los DSML permiten definir dominios CEP y patrones de eventos independientemente del código que los implementa.
- El editor de dominios CEP hace posible la definición gráfica de dominios en los que puede aplicarse CEP, dominios que pueden ser importados, exportados e, incluso, inferidos automáticamente a partir de los eventos que fluyan por una SOA 2.0.
- El editor de patrones de eventos es capaz de reconfigurarse a distintos dominios, así como de generar automáticamente todo el código, que implementa estos patrones, libre de errores y ejecutable en el motor CEP y el ESB.
- Los dos editores desarrollados son amigables e intuitivos, evitándose a los expertos en el negocio la curva de aprendizaje en CEP y SOA 2.0 por la que deberían pasar hasta ser capaces de implementar y desplegar un patrón libre de errores, junto con sus acciones asociadas, en sus sistemas de información.

Por todo lo anterior, se concluye que se han cumplido satisfactoriamente todos los objetivos planteados y que el enfoque dirigido por modelos para CEP en SOA 2.0, propuesto en esta tesis doctoral, pone al alcance de cualquier usuario, sin necesidad de que sea programador ni experto en CEP, la detección de las situaciones críticas o relevantes en las que esté interesado y que personalmente haya definido gráficamente sobre un dominio específico, a partir de la información que fluya en tiempo real por los sistemas complejos y heterogéneos que sean de su interés.

9.2. Líneas de Investigación Futuras

Una vez descritas las conclusiones obtenidas en esta tesis doctoral, se explican las líneas de investigación futuras que se abren con este trabajo.

Recomendación de Patrones de Eventos de Forma Semi-automática

En el Capítulo 8 se han evaluado y discutido todas las aportaciones realizadas en esta tesis doctoral, así como sus principales fortalezas frente a otras propuestas existentes. En concreto, tras un análisis pormenorizado del editor gráfico reconfigurable creado para el

modelado de patrones y su transformación a código, se propone como línea de investigación futura dotarle de la capacidad de recomendar patrones de forma semi-automática, permitiendo al usuario final crear nuevos patrones reutilizando otros patrones de eventos, recomendados por el propio editor a partir del análisis y la interpretación de las estadísticas sobre la ejecución de estos patrones existentes.

Sen y Stojanovic [Sen10a, Sen10b] ya han trabajo en esta línea de investigación. Estos autores determinan que la clave de este enfoque consiste en disponer de una buena representación de los eventos así como de los patrones de eventos con el fin de reconocer cuáles son las relaciones existentes entre estos.

Como continuación de esta tesis doctoral, se pretende trabajar en esta misma línea de investigación pero utilizando el enfoque de desarrollo dirigido por modelos para CEP en SOA 2.0 propuesto en la tesis, así como las tecnologías y los marcos de trabajo basados en Eclipse para la creación de editores de modelado, como EMF, GMF y Epsilon. Así pues, la consecución de este objetivo se llevaría a cabo aunando todas las ventajas, descritas a lo largo de esta disertación, del uso tanto de MDD como de dichas tecnologías y marcos de trabajo.

Generación Dinámica de Patrones de Eventos

En esta tesis doctoral se ha logrado la generación dinámica de dominios CEP, a través de la inferencia de los tipos de eventos que fluyan por una SOA 2.0, creándose automáticamente tanto sus modelos EMF como sus visualizaciones gráficas correspondientes (diagramas); ayudando al experto en el negocio en la definición de estos dominios.

En este sentido, otra de las líneas de investigación que se pretende afrontar en un futuro próximo es la generación dinámica de patrones de eventos a partir de la información, en forma de eventos, que fluya por los sistemas de información de la empresa. Al igual que para la generación dinámica de dominios CEP, la generación dinámica de patrones de eventos será de gran ayuda para el usuario final, que podrá visualizar en el editor de modelado de patrones, creado en esta tesis, los nuevos patrones inferidos automáticamente. Esto hará posible que el usuario no tenga que diseñar los patrones desde cero, sino simplemente realizando algunas modificaciones sobre estos, si así lo estima conveniente. Lo que conllevará un ahorro de tiempo en el modelado de los patrones, más aún cuando se requiera la definición de un gran número de estos para un mismo dominio de aplicación.

Para conseguir este objetivo se hará uso del análisis predictivo, que consiste en la predicción de eventos futuros a partir de los datos históricos que se hayan analizado en el pasado. Entre otros, se aplicarán métodos sofisticados para tal fin como, por ejemplo, el aprendizaje automático.

Aunque se vaticina que esta generación dinámica de patrones distará de ser perfecta, será una aportación muy significativa para el usuario que necesite modelar las situaciones de interés a detectar en su empresa en tiempo real, tratándose de una propuesta encaminada a la realización automática y eficiente de una buena toma de decisiones.

Evaluación del Enfoque Dirigido por Modelos en la Industria

Hasta el momento, el enfoque dirigido por modelos para CEP en SOA 2.0 propuesto y desarrollado en esta tesis doctoral solo se ha evaluado en el mundo de investigación y en el mundo académico.

Como trabajo futuro, se pretende aplicar dicho enfoque al mundo de la industria con la finalidad de comprobar cuál es el nivel de aceptación y satisfacción de los expertos en el negocio que desarrollen su actividad en empresas dedicadas a la fabricación (*manufacturing*), la seguridad en redes o la salud, entre otras. Cabe destacar que durante el desarrollo de esta tesis doctoral, el doctorando ha realizado dos estancias de investigación que, sin duda alguna, serán claves para afrontar esta línea de investigación.

Por un lado, la estancia de tres meses realizada en *Tilburg University* (Holanda), en la que se ha colaborado con el instituto de investigación *European Research Institute in Service Science*, dirigido por el Prof. Dr. Michael P. Papazoglou, supone una vía muy interesante para hacer llegar esta propuesta a empresas de fabricación, debido a las estrechas relaciones existentes entre este instituto, sobre todo el Prof. Papazoglou, con empresas de este sector.

Por otro lado, el doctorando ha realizado una estancia de investigación de un mes en la *University of Applied Sciences Frankfurt am Main* (Alemania) con el grupo de investigación *IT Security, Network Security and Privacy*, dirigido por el Prof. Dr. Martin Kappes. Esta estancia favorecerá la aplicación del enfoque en otros escenarios de seguridad en redes y sistemas de información.

Dada la escalabilidad y la independencia del dominio CEP en el que se aplique el enfoque propuesto, así como el buen grado de funcionalidad y de usabilidad de los editores gráficos de modelado desarrollados en esta tesis doctoral, se espera que la industria reciba con buen agrado este trabajo de investigación.

9.3. Publicaciones

A continuación, se enumeran las principales publicaciones derivadas de esta tesis doctoral, clasificadas en publicaciones internacionales y nacionales.

Publicaciones Internacionales

- J. Boubeta-Puig, G. Ortiz e I. Medina-Bulo. A model-driven approach for facilitating user-friendly design of complex event patterns. En: *Expert Systems with Applications* 41(2) (feb. de 2014), págs. 445-456, (JCR 2012: 1.854).

- J. Boubeta-Puig, G. Ortiz e I. Medina-Bulo. Approaching the Internet of Things through Integrating SOA and Complex Event Processing. en. En: *Handbook of Research on Demand-Driven Web Services: Theory, Technologies, and Applications*. Ed. por Z. Sun y J. Yearwood. IGI Global book series Advances in Web Technologies and Engineering (AWTE). IGI Global, mar. de 2014, págs. 304-323.

- R. Gad, M. Kappes, J. Boubeta-Puig e I. Medina-Bulo. Employing the CEP Paradigm for Network Analysis and Surveillance. En: *Proceedings of The Ninth Advanced International Conference on Telecommunications*. Roma, Italia: IARIA, jun. de 2013, págs. 204-210.
- Y. Taher, J. Boubeta-Puig, W.-J. v. d. Heuvel, G. Ortiz e I. Medina-Bulo. A Model-Driven Approach for Web Service Adaptation Using Complex Event Processing. En: *Advances in Service-Oriented and Cloud Computing*. Ed. por C. Canal y M. Villari. Vol. 393. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2013, págs. 346-359.
- G. Ortiz, J. Boubeta-Puig, A. García-de Prado e I. Medina-Bulo. Towards Event-Driven Context-Aware Web Services. En: *Adaptive Web Services for Modular and Reusable Software Development: Tactics and Solutions*. Ed. por G. Ortiz y J. Cubo. IGI Global, 2013, págs. 148-159.
- J. Cubo, G. Ortiz, J. Boubeta-Puig, H. Foster, y W. Lamersdorf. Preface of the Third International Workshop on Adaptive Services for the Future Internet (WAS4FI 2013). En: *Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC 2013*, vol. 393, C. Canal y M. Villari, Eds. Springer Berlin Heidelberg, 2013.
- R. Gad, J. Boubeta-Puig, M. Kappes e I. Medina-Bulo. Hierarchical events for efficient distributed network analysis and surveillance. En: *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups (WAS4FI-Mashups '12)*. Bertinoro, Italia: ACM, sep. de 2012, págs. 5-11.
- J. Cubo, J. Boubeta-Puig, G. Ortiz, H. Foster, W. Lamersdorf, A. Koschmider, M. Matera, y V. Torres, *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups (WAS4FI-Mashups '12)*, ACM, 2012.
- J. Boubeta-Puig, I. Medina-Bulo, G. Ortiz y G. Fuentes-Landi. Complex event processing applied to early maritime threat detection. En: *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups (WAS4FI-Mashups '12)*. Bertinoro, Italia: ACM, sep. de 2012, págs. 1-4.
- J. Boubeta-Puig, G. Ortiz e I. Medina-Bulo. An Approach of Early Disease Detection using CEP and SOA. En: *Proceedings of The Third International Conferences on Advanced Service Computing*. Roma, Italia: IARIA, sep. de 2011, págs. 143-148.

Publicaciones Nacionales

- J. Boubeta-Puig, G. Ortiz e I. Medina-Bulo. Integración del Internet de las Cosas y las Arquitecturas Orientadas a Servicios: un Caso de Estudio en el Ámbito de

la Domótica. En: *Actas de las IX Jornadas de Ciencia e Ingeniería de Servicios*. Madrid, España, sep. de 2013, págs. 35-49.

- J. Criado, J. Boubeta-Puig, G. Ortiz y L. Iribarne. El lenguaje JET. En: *Desarrollo de software dirigido por modelos: conceptos, métodos y herramientas*. Ed. por J. García, F. García, V. Pelechano, A. Vallecillo, J. M. Vara y C. Vicente-Chicote. Ra-Ma, 2013, págs. 289-306.
- J. Boubeta-Puig, G. Ortiz e I. Medina-Bulo. Integrando el Internet de las Cosas y el Procesamiento de Eventos Complejos en las Arquitecturas Orientadas a Servicios. En: *Actas de las IV Jornadas Predoctorales de la Escuela Superior de Ingeniería*. Cádiz, España, dic. de 2012, págs. 43-46.
- J. A. Dorado-Cerón, J. Boubeta-Puig, G. Ortiz e I. Medina-Bulo. Detección de Ataques de Seguridad mediante la Integración de CEP y SOA 2.0. En: *Actas de las VIII Jornadas de Ciencia e Ingeniería de Servicios*. Almería, España, sep. de 2012, págs. 167-172.
- R. Gad, J. Boubeta-Puig, M. Kappes e I. Medina-Bulo. Leveraging EDA and CEP for Integrating Low-level Network Analysis Methods into Modern, Distributed IT Architectures. En: *Actas de las VIII Jornadas de Ciencia e Ingeniería de Servicios*. Almería, España, sep. de 2012, págs. 13-26.
- J. Boubeta-Puig, G. Ortiz e I. Medina-Bulo. Una propuesta de integración de ESB con un motor CEP. En: *Actas de las III Jornadas Predoctorales de la Escuela Superior de Ingeniería*. Cádiz, España, nov. de 2011, págs. 51-54.
- J. Boubeta-Puig, G. Ortiz e I. Medina-Bulo. Procesamiento de Eventos Complejos en Entornos SOA: Caso de Estudio para la Detección Temprana de Epidemias. En: *Actas de las VII Jornadas de Ciencia e Ingeniería de Servicios*. A Coruña, España: Servizo de publicacions da Universidade da Coruña, sep. de 2011, págs. 63-76.
- J. Boubeta-Puig, G. Ortiz e I. Medina-Bulo. Un Estudio sobre el Procesamiento de Eventos Complejos y su Integración en Arquitecturas Orientadas a Servicios. En: *Actas de las II Jornadas Predoctorales de la Escuela Superior de Ingeniería*. Cádiz, España, nov. de 2010, págs. 57-60.

Capítulo 10

Conclusions and Future Work

«Challenges are gifts that force us to search for a new center of gravity. Don't fight them. Just find a different way to stand.»
(Oprah Winfrey)

This chapter is the English translation for Chapter 9, where the accomplished goals, contributions and conclusions from this thesis are described. Likewise, the future work lines derived from the current research are exposed.

10.1. Contributions and Conclusions

The goals of this thesis, headed towards the model-driven development of domain-specific interfaces for Complex Event Processing (CEP) in Event-Driven Service-Oriented Architectures (ED-SOAs or SOAs 2.0) were set in the first chapter of this dissertation. The main aim bore in mind has been facilitating domain experts both the definition of event patterns to be detected in their information systems and the definition of alerts to be notified in real time and providing the means for doing it in a graphical, user-friendly and intuitive way, hiding implementation details from these users.

In the following lines, the main contributions of this thesis and the conclusions inferred from them are detailed.

State of Art for Existing Approaches for CEP, Graphical Editors and Proposals for the Integration of CEP with EDA and/or SOA

The first objective considered in this thesis was to write a state-of-art report of the existing approaches for CEP, graphical editors for event pattern definition and code generation, as well as proposals for integrating CEP with EDA and/or SOA. This objective has been developed by performing an extensive literature review (see Chapter 2).

Concerning CEP approaches, several works using ontologies for the representation of events and event patterns have been found. Besides, there are also works that propose model-driven approaches in the line of this thesis.

All the analysed approaches have limitations compared to the approach proposed in this thesis. On the one hand, the ontological approaches provide an incomplete set of operator types and data windows. This fact implies a limitation in the expressiveness of complex patterns to be modeled. Furthermore, the way in which some of these ontological models have been structured hinders scalability and understanding of the defined patterns.

On the other hand, the existing model-driven approaches have been classified into two categories —textual or graphical— according to the notation used to represent event patterns. None of them offers the possibility to describe a CEP domain composed of a set of event types together with their domain description and creation date. The lack of these elements in the domain will prevent the chance of sharing the domains among different modeling tools, which might even belong to different users.

Approaches proposing a textual representation for event patterns definition have a lower degree of usability and functionality compared to the one proposed in this thesis. Some of the most significant limitations are the following: 1) a new Domain-Specific Modeling Language (DSML) would be required for each domain where CEP is needed to be applied, causing an increased workload and additional effort for each new language to be implemented; 2) a predefined set of event types for a particular domain —with no option to be modified for other domains— is provided; 3) actions to be executed when detecting situations of interest are not supported.

Furthermore, the approaches making use of graphical notations for event patterns definition also have limitations in terms of functionality and usability, compared to the approach proposed in this thesis. For instance, the analysed approaches have used notations for event patterns definition in table and textual formats, while this thesis has made use of graphical nodes and links. The latest is more intuitive for any user, regardless of his skills concerning CEP technology.

With regard to the existing graphical editors for event pattern definition and code generation, they do not provide a good level of functionality and usability, as discussed in Chapter 8.

Finally, we have not found any proposal for the integration of CEP with SOA 2.0 which provides the benefits of the technological solution proposed in Chapter 6. Existent proposals are mostly dependent on the application domain and do not offer graphical editors for CEP domain and event pattern definition.

Model-Driven Approach for Complex Event Processing in SOA 2.0

The second goal established in this thesis was to propose a model-driven approach that made possible complex event processing in SOA 2.0, minimizing end users learning curve in the technologies required for the detection of relevant or critical situations in real time.

This objective has been fully satisfied through the main contribution of this thesis, that is, a model-driven approach for CEP in SOA 2.0 independent of both the domain

where CEP is applied and the programming languages required to implement not only event patterns but also the actions responsible of notifying the detected situations (see Chapter 3).

This approach consists of two distinct parts. The first part refers to the entire process that is carried out at design time while the second one covers the process that takes place at runtime.

In particular, a CEP domain can be modeled at design time. This model can be automatically inferred from the events flowing through a SOA 2.0 to which the CEP domain graphical editor is connected, or can be manually created by a domain expert using the mentioned editor. From the modeled CEP domain, the user will be able to model patterns about situations of interest to be detected, as well as actions to be performed. Subsequently, these event pattern models will be automatically validated and transformed into code, which will be deployed in a CEP engine and an ESB at runtime; making these tasks transparent to the user thanks to the use of Model-Driven Development (MDD) techniques.

DSML and Graphical Editor for CEP Domain Definition

The third defined goal was to define a DSML and to build a graphical editor for CEP domain definition that would facilitate any user, expert in a particular domain but not in CEP, the description of event types and properties for the domain.

This objective has resulted in the definition of a CEP domain metamodel, along with its restrictions, and with the creation of a graphical editor that supports the modeling of these domains (see Chapter 4). The development method pursued to build this editor follows: 1) metamodel creation using Ecore and Emfatic; 2) generation of the editor initial version using GMF and EuGENia; 3) editor customising by modifying its tool palette and by adding a menu for users to intuitively model CEP domains; 4) implementation of the validation rules, using the Epsilon Validation Language (EVL), in order to check whether a modeled CEP domain is well-formed; and 5) creation of an Eclipse Rich Client Platform (RCP) so that the editor can be run outside Eclipse and on multiple platforms.

One of the most important contributions offered by this DSML is the unification of CEP domain description (event types and properties) by using models, hiding the implementation details necessary to define such domains from domain experts. Furthermore, the built editor makes possible the graphical design and automatic validation of CEP domains, as well as exporting and importing them to be shared and reused by other domain experts.

DSML and Reconfigurable Graphical Editor for Event Pattern Definition and Code Generation

The fourth defined goal was to define a DSML and to build a graphical editor for event pattern definition. The main purpose was providing the users with an intuitive and user-friendly way to describe both the situations to be detected in a particular domain and the actions to be notified to interested users by email or social networking services, among others.

This goal has been successfully completed with the definition of an event pattern metamodel, along with its restrictions, and with the construction of a graphical editor that can be reconfigured for different CEP domains, modeled by domain experts (see Chapter 5). The development method pursued to build this editor follows: 1) metamodel creation using Ecore and Emfatic; 2) generation of the editor initial version using GMF and EuGENia; 3) editor customising by modifying its tool palette and by adding a menu for users to intuitively model event patterns, and also by introducing Java modifications so that the editor can automatically reconfigure from different CEP domain models; 4) implementation of the EVL validation rules in order to check whether a modeled event pattern is well-formed; 5) implementation of model-to-code transformations with Epsilon Generation Language (EGL) for generating code automatically from event pattern models; and 6) creation of a RCP, so that the editor can be run outside Eclipse and on multiple platforms.

One of the most important aspects of this DSML is the possibility of modeling patterns regardless of the implementation. Thanks to the use of MDD every event pattern is graphically designed once and then can be automatically transformed into any particular EPL, such as Esper EPL, StreamSQL or CCL, as well as into any action code to be executed, such as XML. To reach this goal, a graphical editor enables CEP novices to concentrate on modeling the situations to be detected and the actions to be carried out, hiding all implementation details from them. Thus, the obtained and validated pattern models can be exported and imported as well as reused in different information systems through the transformation of these models into both the EPL code required by the chosen CEP engine and the code required by the specific ESB, which is connected to the engine. In this case, the transformation rules for Esper EPL and for XML flows in Mule are provided.

The fact that the editor is able to reconfigure the tool palette dynamically from different CEP domain models will enable user to enjoy a graphical interface adapted to the specific context required. Thereby, this editor brings to reality that the definition of critical or relevant situations in real time by non-technological users.

Technological Solution for the Integration of Complex Event Processing with SOA 2.0

The fifth goal pursued was to integrate CEP with SOA 2.0, as well as with the graphical modeling editors, with the aim of making the proposed model-driven approach a reality.

This goal has been achieved through the design and implementation of a technological solution for integrating CEP with SOA 2.0. An Enterprise Service Bus (ESB) is the key element of this integration that allows us to combine CEP with SOA and EDA. In this scope, the main components that might be integrated in a SOA 2.0 have been described, classified into two categories: event producers —event generators, applications, web services, sensors, Internet of Things (IoT) platforms and social networking services— and event consumers —monitoring consoles, applications, web services, actuators, IoT platforms, social networking and email services and databases.

In addition, the functionalities involved in the integration of CEP with SOA 2.0 have been established: event reception, event transformation, event filtering, automatic gene-

ration of CEP domains, sending events to the CEP engine, adding event patterns to the CEP engine, adding actions for event patterns to the ESB, event pattern detection, complex event reception, and decision-making process (actions to be developed when detected complex events are notified).

Once designed the technological solution, it has been implemented —making use of Mule ESB and Esper CEP engine— and integrated with the graphical modeling editors.

Some of the most remarkable functionalities of this technological solution compared to other existing architectures, are: integration of Mule with Esper, automatic generation of CEP domains, integration with both graphical editors for CEP domain and event pattern definition, as well as the runtime automatic injection of EPL code into the CEP engine and of actions XML code into the ESB. It is important to highlight that all these functionalities do not require users to modify the implementation of the proposed architecture.

Additionally, two case studies have been conducted to demonstrate that this solution can be applied to different application domains. In particular, a case study in the field of home automation and another one in the field of network security have been proposed. As a result, a set of event patterns for the detection of relevant situations in both domains have been obtained.

Conclusions Drawn from the Evaluation

Finally, the sixth defined goal was evaluating the proposed model-driven approach and DSMLs, as well as the graphical editors functionality and usability.

This objective has been achieved through the study and discussion of the feasibility of the proposed model-driven approach and DSMLs, and by evaluating the user satisfaction in terms of functionality and usability of such editors.

The evaluation results allow us to establish the following partial conclusions:

- The DSMLs enable CEP domain and event pattern definition, regardless of the code type in which the patterns have to be implemented.
- The editor for CEP domain definition makes possible the graphical definition of domains in which CEP can be applied. These domain models can be imported, exported, and even automatically inferred from events flowing through a SOA 2.0.
- The editor for event pattern definition and code generation can be reconfigured for different domains. Moreover, the pattern code can be automatically generated from the editor; providing error-free code that can be deployed into CEP engines and ESBs.
- Both editors are user-friendly and intuitive, avoiding experts in business the high learning curve in CEP and SOA 2.0 required for implementing and deploying error-free patterns and actions in their information systems.

Based on the foregoing statements, we conclude that all goals have been successfully met and that the model-driven approach for CEP in SOA 2.0, proposed in this thesis, facilitates

any user, without being programmer nor expert on CEP, the graphical definition of situations of interest for a specific domain and their detection from the real-time information flowing through complex and heterogeneous systems.

10.2. Future Research Lines

Despite the contributions made on this thesis, several future research lines remain open as summarized in the following lines.

Semi-Automatic Event Pattern Recommendation

All contributions made on this thesis together with other proposals main strengths have been evaluated and discussed in Chapter 8. After a detailed analysis of the reconfigurable graphical editor for event pattern definition and code generation, we consider a relevant issue for the future to extend the editor with the ability to recommend event patterns in a semi-automatic way. This will allow end users to create new patterns by reusing other event patterns, which will be suggested by the editor itself from the analysis and interpretation of existing patterns usage statistics.

Sen and Stojanovic [Sen10a, Sen10b] have already worked on this topic. These authors consider that the approach key is having a well-defined event and event pattern representation in order to recognize the relationships between them.

As a continuation of this thesis, it is intended to work on this research line using the proposed model-driven development approach for CEP in SOA 2.0 as well as Eclipse technologies and frameworks for developing modeling editors, such as EMF, GMF and Epsilon. Thus, this goal would be achieved by combining all the advantages provided by the use of MDD techniques and such technologies and frameworks, as it has been mentioned throughout this dissertation.

Dynamic Generation of Event Patterns

Dynamic generation of CEP domains has been achieved in this thesis by inferring the events flowing through a SOA 2.0. As a result, EMF models and the corresponding graphical diagrams can be automatically created; assisting domain experts with CEP domains definition.

In this regard, another research line to be developed in the future is the dynamic generation of event patterns from the information in form of events, which flow through the organization information systems. As in the case of CEP domain dynamic generation, event pattern dynamic generation would be really helpful to end users, since they would be able to visualize the inferred event patterns in the graphical editor. This fact would enable users not to have to design event patterns from scratch, but simply making some modifications on them, saving time when modeling event patterns, particularly when a large number of patterns have to be defined for an application domain.

To reach this goal, we plan to make use of predictive analysis, which consists of predicting future events from historical data analysed in the past. Thereby, sophisticated methods, such as machine learning, will be applied.

Although it is predicted that this dynamic pattern generation will be far from perfection, it will be a very significant contribution for users that require to model situations of interest in their business in real time, facilitating them an automatic and efficient decision-making process.

Model-Driven Approach Evaluation from the Industry

So far, the model-driven approach for CEP in SOA 2.0 proposed and developed in this thesis, has been only evaluated in the research and academic world.

As future work, we intend to apply this approach in the industry with the purpose of checking the acceptance and satisfaction level from business experts, for instance, in the scope of manufacturing, security networks and healthcare, among others. During the development of this dissertation, the doctoral candidate has completed two research stays that will be keys to addressing this research line.

On the one hand, the three-month research stay done at Tilburg University (The Netherlands) in the European Research Institute in Service Science (ERISS), headed by Prof. Dr. Michael P. Papazoglou, could help to bring this proposal to manufacturing companies, thanks to the close relationships existing between this research institute —especially Prof. Papazoglou— and enterprises in this sector.

On the other hand, the doctoral candidate has carried out a one-month stay at the University of Applied Sciences Frankfurt am Main (Germany) in the IT Security, Network Security and Privacy Research Group, led by Prof. Dr. Martin Kappes. This stay will facilitate to deep in the application of the approach in further network and information security scenarios.

Given the scalability and independence of the CEP domain in which the proposed approach is applied, as well as functionality and usability of the graphical editors developed in this thesis, it is expected that the industry receives this research with pleasure.

10.3. Publications

The publications derived from the work in the present thesis are listed below.

International Publications

- J. Boubeta-Puig, G. Ortiz and I. Medina-Bulo. A model-driven approach for facilitating user-friendly design of complex event patterns. In: *Expert Systems with Applications* 41(2) (feb. 2014), pp. 445-456, (JCR 2012: 1.854).

- J. Boubeta-Puig, G. Ortiz and I. Medina-Bulo. Approaching the Internet of Things through Integrating SOA and Complex Event Processing. In: *Handbook of Research*

on Demand-Driven Web Services: Theory, Technologies, and Applications. Ed. Z. Sun and J. Yearwood. IGI Global book series Advances in Web Technologies and Engineering (AWTE). IGI Global, mar. 2014, pp. 304-323.

- R. Gad, M. Kappes, J. Boubeta-Puig and I. Medina-Bulo. Employing the CEP Paradigm for Network Analysis and Surveillance. In: *Proceedings of The Ninth Advanced International Conference on Telecommunications*. Rome, Italy: IARIA, jun. 2013, pp. 204-210.

- Y. Taher, J. Boubeta-Puig, W.-J. v. d. Heuvel, G. Ortiz and I. Medina-Bulo. A Model-Driven Approach for Web Service Adaptation Using Complex Event Processing. In: *Advances in Service-Oriented and Cloud Computing*. Ed. C. Canal and M. Villari. Vol. 393. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2013, pp. 346-359.

- G. Ortiz, J. Boubeta-Puig, A. García-de Prado and I. Medina-Bulo. Towards Event-Driven Context-Aware Web Services. In: *Adaptive Web Services for Modular and Reusable Software Development: Tactics and Solutions*. Ed. G. Ortiz and J. Cubo. IGI Global, 2013, pp. 148-159.

- J. Cubo, G. Ortiz, J. Boubeta-Puig, H. Foster, and W. Lamersdorf. Preface of the Third International Workshop on Adaptive Services for the Future Internet (WAS4FI 2013). In: *Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC 2013*, vol. 393, C. Canal and M. Villari, Eds. Springer Berlin Heidelberg, 2013.

- R. Gad, J. Boubeta-Puig, M. Kappes and I. Medina-Bulo. Hierarchical events for efficient distributed network analysis and surveillance. In: *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups (WAS4FI-Mashups '12)*. Bertinoro, Italy: ACM, sep. 2012, pp. 5-11.

- J. Cubo, J. Boubeta-Puig, G. Ortiz, H. Foster, W. Lamersdorf, A. Koschmider, M. Matera, and V. Torres, *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups (WAS4FI-Mashups '12)*, ACM, 2012.

- J. Boubeta-Puig, I. Medina-Bulo, G. Ortiz and G. Fuentes-Landi. Complex event processing applied to early maritime threat detection. In: *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups (WAS4FI-Mashups '12)*. Bertinoro, Italy: ACM, sep. 2012, pp. 1-4.

- J. Boubeta-Puig, G. Ortiz and I. Medina-Bulo. An Approach of Early Disease Detection using CEP and SOA. In: *Proceedings of The Third International Conferences on Advanced Service Computing*. Rome, Italy: IARIA, sep. 2011, pp. 143-148.

National Publications

- J. Boubeta-Puig, G. Ortiz and I. Medina-Bulo. Integración del Internet de las Cosas y las Arquitecturas Orientadas a Servicios: un Caso de Estudio en el Ámbito de la Domótica. In: *Actas de las IX Jornadas de Ciencia e Ingeniería de Servicios*. Madrid, Spain, sep. 2013, pp. 35-49.

- J. Criado, J. Boubeta-Puig, G. Ortiz and L. Iribarne. El lenguaje JET. In: *Desarrollo de software dirigido por modelos: conceptos, métodos y herramientas*. Ed. J. García, F. García, V. Pelechano, A. Vallecillo, J. M. Vara and C. Vicente-Chicote. Ra-Ma, 2013, pp. 289-306.

- J. Boubeta-Puig, G. Ortiz e I. Medina-Bulo. Integrando el Internet de las Cosas y el Procesamiento de Eventos Complejos en las Arquitecturas Orientadas a Servicios. In: *Actas de las IV Jornadas Predoctorales de la Escuela Superior de Ingeniería*. Cádiz, Spain, dec. 2012, pp. 43-46.

- J. A. Dorado-Cerón, J. Boubeta-Puig, G. Ortiz and I. Medina-Bulo. Detección de Ataques de Seguridad mediante la Integración de CEP y SOA 2.0. In: *Actas de las VIII Jornadas de Ciencia e Ingeniería de Servicios*. Almería, Spain, sep. 2012, pp. 167-172.

- R. Gad, J. Boubeta-Puig, M. Kappes and I. Medina-Bulo. Leveraging EDA and CEP for Integrating Low-level Network Analysis Methods into Modern, Distributed IT Architectures. In: *Actas de las VIII Jornadas de Ciencia e Ingeniería de Servicios*. Almería, Spain, sep. 2012, pp. 13-26.

- J. Boubeta-Puig, G. Ortiz and I. Medina-Bulo. Una propuesta de integración de ESB con un motor CEP. In: *Actas de las III Jornadas Predoctorales de la Escuela Superior de Ingeniería*. Cádiz, Spain, nov. 2011, pp. 51-54.

- J. Boubeta-Puig, G. Ortiz and I. Medina-Bulo. Procesamiento de Eventos Complejos en Entornos SOA: Caso de Estudio para la Detección Temprana de Epidemias. In: *Actas de las VII Jornadas de Ciencia e Ingeniería de Servicios*. A Coruña, Spain: Servizo de publicacions da Universidade da Coruña, sep. 2011, pp. 63-76.

- J. Boubeta-Puig, G. Ortiz and I. Medina-Bulo. Un Estudio sobre el Procesamiento de Eventos Complejos y su Integración en Arquitecturas Orientadas a Servicios. In: *Actas de las II Jornadas Predoctorales de la Escuela Superior de Ingeniería*. Cádiz, Spain, nov. 2010, pp. 57-60.

Lista de Acrónimos

ALERT Active support and reaL-time coordination based on Event pRocessing in FLOSS developmenT

ANR L'Agence Nationale de la Recherche

API Application Programming Interface

ARP Address Resolution Protocol

BEMN Business Event Modeling Notation

BPEL Business Process Execution Language

CCL Continuous Computation Language

CEP Complex Event Processing

COMPAS Compliance-driven Models, Languages, and Architectures for Services

CQL Continuous Query Language

CSV Comma-Separated Values

DS-EPL Domain-Specific Event Processing Language

DSML Domain-Specific Modeling Language

ECA Event-Condition-Action

EDA Event-Driven Architecture

ED-SOA Event-Driven Service-Oriented Architecture

EGL Epsilon Generation Language

ELE ETALIS Language for Events

EMF Eclipse Modeling Framework

EMOF Essential MOF

EOL Epsilon Object Language

EPL Event Processing Language

EP-SPARQL Event Processing SPARQL

ESB Enterprise Service Bus

ETALIS Event TrAnsaction Logic Inference System

ETL Epsilon Transformation Language

EVL Epsilon Validation Language

FP7 Framework Programme 7

GMF Eclipse Graphical Modeling Framework

GWT Google Web Toolkit

HTTP HyperText Transfer Protocol

ICMP Internet Control Message Protocol

IDE Integrated Development Environment

IoT Internet of Things

IT Information Technology

IP Internet Protocol

JMS Java Message Service

JSON JavaScript Object Notation

LOC Lines of Code

MAC Media Access Control

MDD Model-Driven Development

MOF Meta-Object Facility

NoSQL Not only SQL

OASIS Organization for the Advancement of Structured Information Standards

OCL Object Constraint Language

OMG Object Management Group

OWL Web Ontology Language

RCP Rich Client Platform

RGB Red Green Blue

RDF Resource Description Framework

RDFS Resource Description Framework Schema

REST REpresentational State Transfer

RFID Radio Frequency IDentification

SARI Sense And Respond Infrastructure

SCA Service Component Architecture

SOA Service-Oriented Architecture

SOAP Simple Object Access Protocol

SocEDA SOCial Event Driven Architecture

SQL Structured Query Language

SVG Scalable Vector Graphics

TCP Transmission Control Protocol

UCA Universidad de Cádiz

UDP User Datagram Protocol

UML Unified Modeling Language

URL Uniform Resource Locator

VM Virtual Machine

XML eXtensible Markup Language

XMI XML Metadata Interchange

Bibliografía

[AG14] Software AG. Apama analytics & decisions platform. `http://www.softwareag.com/corporate/products/bigdata/apama_analytics/overview/`, 2014. [último acceso: 01-06-2014].

[ALE13] ALERT. Active support and reaL-time coordination based on event pRocessing in FLOSS developmenT. `http://www.alert-project.eu/`, 2013. [último acceso: 01-06-2014].

[Ani14] Darko Anicic y Paul Fodor. ETALIS - event-driven transaction logic inference system. `https://code.google.com/p/etalis/`, 2014. [último acceso: 01-06-2014].

[Apa14a] Apache. ActiveMQ. `http://activemq.apache.org/`, 2014. [último acceso: 01-06-2014].

[Apa14b] Apache. ODE. `http://ode.apache.org/`, 2014. [último acceso: 01-06-2014].

[Apa14c] Apache. ServiceMix ESB. `http://servicemix.apache.org/`, 2014. [último acceso: 01-06-2014].

[Arm10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica y Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):páginas 50–58, abril 2010.

[Atk01] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Peach, Jurgen Wust y Jorg Zettel. *Component-Based Product Line Engineering with UML*. Addison-Wesley Professional, Nueva York, Estados Unidos, 1 edición, noviembre 2001. ISBN 9780201737912.

[Atz10] Luigi Atzori, Antonio Iera y Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):páginas 2787–2805, octubre 2010.

[BEA07] BEA. Event processing market pulse 2007, 2007. [último acceso: 01-06-2014].

[Blo14] The Complex Event Processing Blog. `http://www.thecepblog.com/`, 2014. [último acceso: 01-06-2014].

[Bo08] Deng Bo, Ding Kun y Zhang Xiaoyi. A high performance enterprise service bus platform for complex event processing. En *Seventh International Conference on Grid and Cooperative Computing (GCC)*, páginas 577–582. octubre 2008.

[BP11] Juan Boubeta-Puig, Guadalupe Ortiz y Inmaculada Medina-Bulo. An approach of early disease detection using CEP and SOA. En *Proceedings of The Third International Conferences on Advanced Service Computing*, páginas 143–148. IARIA, Roma, Italia, septiembre 2011.

[BP12] Juan Boubeta-Puig, Inmaculada Medina-Bulo, Guadalupe Ortiz y Germán Fuentes-Landi. Complex event processing applied to early maritime threat detection. En *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups*, WAS4FI-Mashups '12, páginas 1–4. ACM, Bertinoro, Italia, septiembre 2012.

[BP14a] Juan Boubeta-Puig, Guadalupe Ortiz y Inmaculada Medina-Bulo. Approaching the Internet of Things through integrating SOA and complex event processing. En Zhaohao Sun y John Yearwood, editores, *Handbook of Research on Demand-Driven Web Services: Theory, Technologies, and Applications*, IGI Global book series Advances in Web Technologies and Engineering (AWTE), páginas 304–323. IGI Global, marzo 2014.

[BP14b] Juan Boubeta-Puig, Guadalupe Ortiz y Inmaculada Medina-Bulo. A model-driven approach for facilitating user-friendly design of complex event patterns. *Expert Systems with Applications*, 41(2):páginas 445–456, febrero 2014.

[Bru14] Ralf Bruns, Jürgen Dunkel, Stefan Lier y Henrik Masbruch. DS-EPL: domain-specific event processing language. En *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, páginas 83–94. ACM, Nueva York, Estados Unidos, 2014.

[Cat11] Rick Cattell. Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, 39(4):páginas 12–27, mayo 2011.

[Cha94] S. Chakravarthy y D. Mishra. Snoop: an expressive event specification language for active databases. *Data & Knowledge Engineering*, 14(1):páginas 1–26, 1994.

[Cha09] Sharma Chakravarthy y Qingchun Jiang. *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*. Springer Publishing Company, Incorporated, 1 edición, 2009. ISBN 9780387710020.

[Cha10] K. Mani Chandy y W. Roy Schulte. *Event Processing: Designing IT Systems for Agile Companies*. McGraw-Hill, Estados Unidos, 2010. ISBN 9780071633505.

[Cha11] Surajit Chaudhuri, Umeshwar Dayal y Vivek Narasayya. An overview of business intelligence technology. *Communications of the ACM*, 54(8):páginas 88–98, agosto 2011.

[Chr08] Binildas A. Christudas. *Service Oriented Java Business Integration: Enterprise Service Bus integration solutions for Java developers*. Packt Publishing, Birmingham, 1 edición, 2008. ISBN 9781847194404.

[Com14] OpenESB Community. OpenESB. `http://www.open-esb.net/index.php/openesb-resources/openesb-documentation`, 2014. [último acceso: 01-06-2014].

[CSA14] OASIS Open CSA. Service component architecture (SCA). `http://www.oasis-opencsa.org/sca`, 2014. [último acceso: 01-06-2014].

[Cug12] Gianpaolo Cugola y Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 44(3):páginas 1–62, junio 2012.

[Dag10] Schloss Dagstuhl. The event processing manifesto. Dagstuhl seminar proceedings 10201, mayo 2010.

[Dav09] Jeff Davis. *Open Source SOA*. Manning Publications, 2009. ISBN 978-1-933988-54-2.

[DC12] José Antonio Dorado-Cerón, Juan Boubeta-Puig, Guadalupe Ortiz y Inmaculada Medina-Bulo. Detección de ataques de seguridad mediante la integración de CEP y SOA 2.0. En *Actas de las VIII Jornadas de Ciencia e Ingeniería de Servicios*, páginas 167–172. Almería, España, septiembre 2012.

[Dec07] Gero Decker, Alexander Grosskopf y Alistair Barros. A graphical notation for modeling complex events in business processes. En *11th IEEE International Enterprise Distributed Object Computing Conference*, páginas 27–36. Annapolis, MD, octubre 2007.

[Dos14] David Dossot, John D'Emic y Victor Romero. *Mule in Action*. Manning Publications, Nueva York, Estados Unidos, 2 edición, 2014. ISBN 9781617290824.

[Dun11] Jürgen Dunkel, Alberto Fernández, Rubén Ortiz y Sascha Ossowski. Event-driven architecture for decision support in traffic management systems. *Expert Systems with Applications*, 38(6):páginas 6530–6539, junio 2011.

[Ecl12] Eclipse Foundation. Emfatic. `http://www.eclipse.org/modeling/emft/emfatic/`, 2012. [último acceso: 01-06-2014].

[Ecl14a] Eclipse Foundation. Eclipse. http://eclipse.org/, 2014. [último acceso: 01-06-2014].

[Ecl14b] Eclipse Foundation. Eclipse modeling framework. http://www.eclipse.org/modeling/emf/, 2014. [último acceso: 01-06-2014].

[Ecl14c] Eclipse Foundation. Epsilon. http://www.eclipse.org/epsilon/, 2014. [último acceso: 01-06-2014].

[Ecl14d] Eclipse Foundation. Epsilon generation language. http://www.eclipse.org/epsilon/doc/egl/, 2014. [último acceso: 01-06-2014].

[Ecl14e] Eclipse Foundation. Epsilon object language. http://www.eclipse.org/epsilon/doc/eol/, 2014. [último acceso: 01-06-2014].

[Ecl14f] Eclipse Foundation. Epsilon transformation language. http://www.eclipse.org/epsilon/doc/etl/, 2014. [último acceso: 01-06-2014].

[Ecl14g] Eclipse Foundation. Epsilon validation language. http://www.eclipse.org/epsilon/doc/evl/, 2014. [último acceso: 01-06-2014].

[Ecl14h] Eclipse Foundation. GEF: graphical editing framework. http://www.eclipse.org/gef/, 2014. [último acceso: 01-06-2014].

[Ecl14i] Eclipse Foundation. GMP: graphical modeling project. http://www.eclipse.org/modeling/gmp/, 2014. [último acceso: 01-06-2014].

[Ecl14j] Eclipse Foundation. Xtext. http://www.eclipse.org/Xtext/, 2014. [último acceso: 01-06-2014].

[Esp14] EsperTech. Esper - complex event processing. http://esper.codehaus.org/, 2014. [último acceso: 01-06-2014].

[Etz10] Opher Etzion y Peter Niblett. *Event Processing in Action*. Manning, Stamford, Estados Unidos, 2010. ISBN 9781935182214.

[Etz13] Opher Etzion y Barbara von Halle. ER 2013 tutorial: modeling the event driven world. http://www.slideshare.net/opher.etzion/er-2013-tutorial-modeling-the-event-driven-world, 2013. [último acceso: 01-06-2014].

[Eve10] Event Processing Technical Society. Event processing glossary - version 2.0. http://www.complexevents.com/wp-content/uploads/2011/08/EPTS_Event_Processing_Glossary_v2.pdf, 2010. [último acceso: 01-06-2014].

[Fai06] Ted Faison. *Event-Based Programming: Taking Events to the Limit*. Apress, Nueva York, Estados Unidos, 1 edición, abril 2006. ISBN 9781590596432.

[Fow10] Martin Fowler y Rebecca Parsons. *Domain Specific Languages*. Addison Wesley, 1 edición, septiembre 2010. ISBN 9780321712943.

[Gad12a] Ruediger Gad, Juan Boubeta-Puig, Martin Kappes y Inmaculada Medina-Bulo. Hierarchical events for efficient distributed network analysis and surveillance. En *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups*, WAS4FI-Mashups '12, páginas 5–11. ACM, Bertinoro, Italia, septiembre 2012.

[Gad12b] Ruediger Gad, Juan Boubeta-Puig, Martin Kappes y Inmaculada Medina-Bulo. Leveraging EDA and CEP for integrating low-level network analysis methods into modern, distributed IT architectures. En *Actas de las VIII Jornadas de Ciencia e Ingeniería de Servicios*, páginas 13–26. Almería, España, septiembre 2012.

[Gad13] Ruediger Gad, Martin Kappes, Juan Boubeta-Puig y Inmaculada Medina-Bulo. Employing the CEP paradigm for network analysis and surveillance. En *Proceedings of The Ninth Advanced International Conference on Telecommunications*, páginas 204–210. IARIA, Roma, Italia, junio 2013.

[GM13a] Jesús García-Molina. Desarrollo digirido por modelos: un nuevo paradigma de construcción de software. En Jesús García, Félix Óscar García, Vicente Pelechano, Antonio Vallecillo, Juan Manuel Vara y Cristina Vicente-Chicote, editores, *Desarrollo de software dirigido por modelos: conceptos, métodos y herramientas*, páginas 289–306. Ra-Ma, 2013. ISBN 978-84-9964-215-4.

[GM13b] Jesús García-Molina. Una introducción al metamodelo. En Jesús García, Félix Óscar García, Vicente Pelechano, Antonio Vallecillo, Juan Manuel Vara y Cristina Vicente-Chicote, editores, *Desarrollo de software dirigido por modelos: conceptos, métodos y herramientas*, páginas 289–306. Ra-Ma, 2013. ISBN 978-84-9964-215-4.

[Gén13] Gonzalo Génova. Conceptos básicos de modelado. En Jesús García, Félix Óscar García, Vicente Pelechano, Antonio Vallecillo, Juan Manuel Vara y Cristina Vicente-Chicote, editores, *Desarrollo de software dirigido por modelos: conceptos, métodos y herramientas*, páginas 289–306. Ra-Ma, 2013. ISBN 978-84-9964-215-4.

[Goo14] Google. Google web toolkit. `http://www.gwtproject.org/`, 2014. [último acceso: 01-06-2014].

[GP08] Cesar Gonzalez-Perez y Brian Henderson-Sellers. *Metamodelling for Software Engineering*. Wiley, Chichester, UK; Hoboken, NJ, 1 edición, octubre 2008. ISBN 9780470030363.

[Gro09] Richard C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language*. Addison-Wesley Professional, Upper Saddle River, NJ, 1 edición, marzo 2009. ISBN 9780321534071.

[Gub13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic y Marimuthu Palaniswami. Internet of Things (IoT): a vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):páginas 1645–1660, septiembre 2013.

[Han13] Dain Hansen. Oracle fast data: Real-time strategies for big data and business analytics. An Oracle white paper, Oracle, marzo 2013.

[He08] Meng He, Zhao Zheng, Guixiang Xue y Xiaoming Du. Event driven RFID based exhaust gas detection services oriented system research. En *4th International Conference on Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08*, páginas 1–4. Dalian, China, octubre 2008.

[Hic14] Rich Hickey. Clojure. `http://clojure.org/`, 2014. [último acceso: 01-06-2014].

[IBM14] IBM. Operational decision manager. `http://www-03.ibm.com/software/products/en/odm`, 2014. [último acceso: 01-06-2014].

[Jac14] Jackson. FasterXML/jackson. `https://github.com/FasterXML/jackson`, 2014. [último acceso: 01-06-2014].

[JBo13a] JBoss. JBoss ESB - JBoss community. `http://jboss.org/jbossesb`, 2013. [último acceso: 01-06-2014].

[JBo13b] JBoss Community. Drools. `http://www.jboss.org/drools`, 2013. [último acceso: 01-06-2014].

[JBo14] JBoss Community. Drools fusion. `http://www.jboss.org/drools/drools-fusion.html`, 2014. [último acceso: 01-06-2014].

[JR13] Álvaro Jiménez Rielo, David Granada y Antonio García Domínguez. EuGENia. En *Desarrollo de software dirigido por modelos: conceptos, métodos y herramientas*, páginas 155–171. Ra-Ma, Madrid, España, 1 edición, 2013. ISBN 978-84-9964-215-4.

[JSO14] JSON. Introducing JSON, 2014. [último acceso: 01-06-2014].

[Jur10] Matjaz B. Juric. WSDL and BPEL extensions for event driven architecture. *Information and Software Technology*, 52(10):páginas 1023–1043, octubre 2010.

[Kav10] Albert Kavelar, Hannes Obweger, Josef Schiefer y Martin Suntinger. Web-based decision making for complex event processing systems. En *IEEE Congress on Services*, páginas 453–458. IEEE Computer Society, Los Alamitos, CA, Estados Unidos, 2010.

[Kle03] Anneke Kleppe, Jos Warmer y Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional, Boston, 1 edición, 2003. ISBN 9780321194428.

[Kol10] Dimitrios S. Kolovos, Louis M. Rose, Saad Bin Abid, Richard F. Paige, Fiona A. C. Polack y Goetz Botterweck. Taming EMF and GMF using model transformation. En Dorina C. Petriu, Nicolas Rouquette y Oystein Haugen, editores, *Model Driven Engineering Languages and Systems*, tomo 6394 de *Lecture Notes in Computer Science*, páginas 211–225. Springer Berlin Heidelberg, Berlin, Alemania, 2010.

[Kol14] Dimitris Kolovos, Louis Rose, Antonio García-Domínguez y Richard Paige. *The Epsilon Book*. 2014.

[Lev09] O. Levina y V. Stantchev. Realizing event-driven SOA. En *Fourth International Conference on Internet and Web Applications and Services, 2009. ICIW '09*, páginas 37–42. mayo 2009.

[Log14] LogMeIn. Xively - business solutions for the Internet of Things. `https://xively.com/`, 2014. [último acceso: 01-06-2014].

[Luc02] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, MA, Estados Unidos, 2002. ISBN 0201727897.

[Luc12] David Luckham. *Event Processing for Business: Organizing the Real-Time Enterprise*. Wiley, Nueva Jersey, Estados Unidos, 2012. ISBN 978-0-470-53485-4.

[Mar06] Jean-Louis Maréchaux. Combining service-oriented architecture and event-driven architecture using an enterprise service bus. Informe técnico, IBM, 2006.

[Müh06] Gero Mühl, Ludger Fiege y Peter Pietzuch. *Distributed Event-Based Systems*. Springer, Berlin, 2006. ISBN 9783540326519.

[Mic06] Brenda Michelson. Event-driven architecture overview: Event-driven SOA is just part of the EDA story. `http://www.customers.com/articles/event-driven-architecture-overview/`, febrero 2006. [último acceso: 01-06-2014].

[Mic14] Microsoft. Windows workflow foundation. `http://msdn.microsoft.com/en-us/library/vstudio/ms735967%28v=vs.90%29.aspx`, 2014. [último acceso: 01-06-2014].

[Mul13] Emmanuel Mulo, Uwe Zdun y Schahram Dustdar. Domain-specific language for event-based compliance monitoring in process-driven SOAs. *Service Oriented Computing and Applications*, 7(1):páginas 59–73, marzo 2013.

[Mul14] MuleSoft. Mule ESB. `http://www.mulesoft.org/`, 2014. [último acceso: 01-06-2014].

[OAS07] OASIS. Web service business process execution language (WS-BPEL) 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`, abril 2007. [último acceso: 01-06-2014].

[Obw10] Hannes Obweger, Josef Schiefer, Peter Kepplinger y Martin Suntinger. Discovering hierarchical patterns in event-based systems. En *IEEE International Conference on Services Computing (SCC)*, páginas 329–336. Miami, FL, Estados Unidos, julio 2010.

[Obw11] Hannes Obweger, Josef Schiefer, Martin Suntinger y Peter Kepplinger. Model-driven rule composition for event-based systems. *International Journal of Business Process Integration and Management*, 5(4):páginas 344–357, enero 2011.

[OMG14a] OMG. MetaObject facility. `http://www.omg.org/mof/`, 2014. [último acceso: 01-06-2014].

[OMG14b] OMG. Object constraint language. `http://www.omg.org/spec/OCL/`, 2014. [último acceso: 01-06-2014].

[OMG14c] OMG. Unified modeling language. `http://www.uml.org/`, 2014. [último acceso: 01-06-2014].

[OMG14d] OMG. XML metadata interchange. `http://www.omg.org/spec/XMI/`, 2014. [último acceso: 01-06-2014].

[Ora14] Oracle. Oracle event processing. `http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/index.html`, 2014. [último acceso: 01-06-2014].

[Ort07] Guadalupe Ortiz. *Integrating Extra-Functional Properties in Model-Driven Web Service Development.* Tesis Doctoral, abril 2007.

[Ort13] Guadalupe Ortiz, Juan Boubeta-Puig, Alfonso García-de Prado y Inmaculada Medina-Bulo. Towards event-driven context-aware web services. En Guadalupe Ortiz y Javier Cubo, editores, *Adaptive Web Services for Modular and Reusable Software Development: Tactics and Solutions*, páginas 148–159. IGI Global, 2013. ISBN 978-1-4666-2089-6.

[Pap06] Michael P. Papazoglou y Willem-Jan van den Heuvel. Service-oriented design and development methodology. *International Journal of Web Engineering and Technology*, 2(4):páginas 412–442, 2006.

[Pap07a] Michael P. Papazoglou. *Web Services: Principles and Technology.* Prentice Hall, 1 edición, septiembre 2007. ISBN 9780321155559.

[Pap07b] Michael P. Papazoglou y Willem-Jan van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):páginas 389–415, julio 2007.

[Pas09] Adrian Paschke. A semantic design pattern language for complex event processing. En *Intelligent Event Processing, Papers from the 2009 AAAI Spring Symposium*, páginas 54–60. 2009.

[Pas12] Adrian Paschke, Harold Boley, Zhili Zhao, Kia Teymourian y Tara Athan. Reaction RuleML 1.0: Standardized semantic reaction rules. En Antonis Bikakis y Adrian Giurca, editores, *Rules on the Web: Research and Applications*, tomo 7438 de *Lecture Notes in Computer Science*, páginas 100–119. Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-32688-2.

[Pet14] Petalslink. Petals ESB. `https://research.linagora.com/display/distribution/ESB`, 2014. [último acceso: 01-06-2014].

[Pro] Real Time Intelligence & Complex Event Processing. `http://www.complexevents.com/`. [último acceso: 01-06-2014].

[Rad09] Tijs Rademakers y Jos Dirksen. *Open-Source ESBs in Action*. Manning, Greenwich, 2009. ISBN 1933988215.

[Rav10] Dominique Ravier. A service oriented framework architecture for intelligent video surveillance systems. En *Fifth International Conference on Digital Telecommunications (ICDT)*, páginas 123–127. 2010.

[Rie14] Stefan Ried. The Forrester Wave: Hybrid 2 integration, Q1 2014. Informe técnico, Forrester Research, Estados Unidos, febrero 2014.

[Rom11] Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy y Frank Eliassen. The DigiHome service-oriented platform. *Software: Practice and Experience*, 43(10):páginas 1205–1218, octubre 2011.

[Roz09] Szabolcs Rozsnyai, Josef Schiefer y Heinz Roth. SARI-SQL: event query language for event analysis. En *2009 IEEE Conference on Commerce and Enterprise Computing*, páginas 24–32. Viena, Austria, julio 2009.

[Rus11] Philip Russom. Big data analytics. Informe técnico, Data Warehousing Institute, 2011.

[Sch05] Josef Schiefer y Andreas Seufert. Management and controlling of time-sensitive business processes with sense & respond. En *International Conference on Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, tomo 1, páginas 77–82. noviembre 2005.

[Sen09] Sinan Sen, Nenad Stojanovic y Ruofeng Lin. A graphical editor for complex event pattern generation. En *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, páginas 1–2. ACM, Nueva York, Estados Unidos, 2009.

[Sen10a] Sinan Sen y Nenad Stojanovic. GRUVe: a methodology for complex event pattern life cycle management. En Barbara Pernici, editor, *Advanced Information Systems Engineering*, tomo 6051 de *Lecture Notes in Computer Science*, páginas 209–223. Springer Berlin Heidelberg, 2010.

[Sen10b] Sinan Sen, Nenad Stojanovic y Ljiljana Stojanovic. An approach for iterative event pattern recommendation. En *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems*, DEBS '10, páginas 196–205. ACM, Nueva York, Estados Unidos, julio 2010.

[Sen12] Sinan Sen, Ljiljana Stojanovic y Ivan Obradovic. Interaction pattern editor. Informe técnico, ALERT, abril 2012.

[Soc13a] SocEDA. CEP editor. `http://research.petalslink.org/display/soceda/CEP+Editor`, 2013. [último acceso: 01-06-2014].

[Soc13b] SocEDA. SOCial Event Driven Architecture. `https://research.linagora.com/display/soceda/SocEDA+Overview`, 2013. [último acceso: 01-06-2014].

[Sos11] Barrie Sosinsky. *Cloud Computing Bible*. Wiley, Indiana, Estados Unidos, 2011. ISBN 978-0-470-90356-8.

[Sot09] Davide Sottara, Alberto Manservisi, Paola Mello, Gabriele Colombini y Luca Luccarini. A CEP-based SOA for the management of WasteWater treatment plants. En *IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS)*, páginas 58–65. Crema, Italia, septiembre 2009.

[Stü09] Roland Stühmer, Darko Anicic, Sinan Sen, Jun Ma, Kay-Uwe Schmidt y Nenad Stojanovic. Lifting events in RDF from interactions with annotated web pages. En Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta y Krishnaprasad Thirunarayan, editores, *The Semantic Web - ISWC 2009*, número 5823 en Lecture Notes in Computer Science, páginas 893–908. Springer Berlin Heidelberg, enero 2009. ISBN 978-3-642-04930-9.

[Sta06] Thomas Stahl, Markus Voelter y Krzysztof Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006. ISBN 0470025700.

[Ste08] Dave Steinberg, Frank Budinsky, Marcelo Paternostro y Ed Merks. *EMF: Eclipse Modeling Framework*. Eclipse Series. Addison-Wesley Professional, 2 edición, 2008. ISBN 978-0321331885.

[Sto10] Michael Stonebraker. SQL databases v. NoSQL databases. *Communications of the ACM*, 53(4):páginas 10–11, abril 2010.

[Str14] StreamBase. StreamBase studio. `www.streambase.com/products/streambasecep/streambase-studio`, 2014. [último acceso: 01-06-2014].

[Syb14] Sybase. SAP sybase event stream processor. `http://www.sybase.com/products/financialservicessolutions/complex-event-processing`, 2014. [último acceso: 01-06-2014].

[Tay08] Hugh Taylor, Yochem Yochem, Les Phillips y Frank Martinez. *Event-driven architecture: how SOA enables the real-time enterprise*. Addison-Wesley, Boston, Mass.; Londres, 2008. ISBN 9780321322111.

[Tec14] Sly Technologies. jNetPcap OpenSource. `http://jnetpcap.com/`, 2014. [último acceso: 01-06-2014].

[Thi14] ThingSpeak. Internet of Things. `https://www.thingspeak.com/`, 2014. [último acceso: 01-06-2014].

[TIB13] TIBCO. StreamBase studio. `http://marketing.streambase.com/acton/form/1262/004b:d-0001/0/index.htm`, 2013. [último acceso: 01-06-2014].

[Tsu12] Satoshi Tsuchiya, Yoshinori Sakamoto, Yuichi Tsuchimoto y Vivian Lee. Big data processing in cloud environments. *Fujitsu Scientific and Technical Journal*, 48(2):páginas 159–168, 2012.

[Twi14] Twitter. `https://twitter.com/`, 2014. [último acceso: 01-06-2014].

[Uhm11] Yoonsik Uhm, Minsoo Lee, Zion Hwang, Yong Kim y Sehyun Park. A multi-resolution agent for service-oriented situations in ubiquitous domains. *Expert Systems with Applications*, 38(10):páginas 13.291–13.300, septiembre 2011.

[Vid11] Kresimir Vidackovic y Anette Weisbecker. A methodology for dynamic service compositions based on an event-driven approach. En *SRII Global Conference, 2011 Annual*, páginas 484–494. marzo 2011.

[Vik13] Konstantin Vikhorev, Richard Greenough y Neil Brown. An advanced energy management framework to promote energy awareness. *Journal of Cleaner Production*, 43:páginas 103–112, marzo 2013.

[Vog13] Lars Vogel. *Eclipse 4 RCP: The complete guide to Eclipse application development*. Lars Vogel, 2013. ISBN 9783943747072.

[W3C14a] W3C. OWL web ontology language. `http://www.w3.org/TR/owl-features/`, 2014. [último acceso: 01-06-2014].

[W3C14b] W3C. RDF schema 1.1. `http://www.w3.org/TR/rdf-schema/`, 2014. [último acceso: 01-06-2014].

[Wie09] Matthias Wieland, Daniel Martin, Oliver Kopp y Frank Leymann. SOEDA: a method for specification and implementation of applications on a service-oriented event-driven architecture. En Witold Abramowicz, editor, *Business Information Systems*, número 21 en Lecture Notes in Business Information Processing, páginas 193–204. Springer Berlin Heidelberg, enero 2009. ISBN 978-3-642-01189-4.

[Yao11] Wen Yao, Chao-Hsien Chu y Zang Li. Leveraging complex event processing for smart hospitals using RFID. *Journal of Network and Computer Applications*, 34(3):páginas 799–810, mayo 2011.

[Yua09] Soe-Tsyr Yuan y Mei-Rung Lu. An value-centric event driven model and architecture: A case study of adaptive complement of SOA for distributed care service delivery. *Expert Systems with Applications*, 36(2):páginas 3671–3694, marzo 2009.

[Zan08] Chuanzhen Zang, Yushun Fan y Renjing Liu. Architecture, implementation and application of complex event processing in enterprise information systems based on RFID. *Information Systems Frontiers*, 10(5):páginas 543–553, noviembre 2008.

[Zmu10] Daniel Zmuda, Marek Psiuk y Krzysztof Zielinski. Dynamic monitoring framework for the SOA execution environment. *Procedia Computer Science*, 1(1):páginas 125–133, mayo 2010.

Anexo A

Cuestionario de Evaluación del Editor Gráfico de Patrones de Eventos

- Nombre y apellidos:
- Profesión (estudiante o investigador):
- Universidad:

1. ¿Se considera un experto en el procesamiento de eventos complejos (CEP)?

   a) No.
   b) Sí.

   En caso afirmativo, ¿qué motor o motores CEP ha utilizado y con qué propósito?

2. ¿Es experto en el dominio para el que se van a definir los patrones de eventos?

   a) No.
   b) Sí.

3. ¿Cómo le gustaría definir los patrones de eventos?

   a) Utilizando un editor gráfico.
   b) Codificando el patrón de eventos usando un lenguaje de programación.
   c) Utilizando un editor gráfico y también codificándolo usando un lenguaje de programación.

4. ¿Qué tipo de usuario es el más adecuado para utilizar el editor de patrones de eventos que está evaluando (el editor, de aquí en adelante)?

a) Usuarios del dominio de aplicación (sin necesidad de conocimientos de programación).

b) Programadores.

c) No lo sé.

d) Otros (especifíquelos):

5. ¿Está claro cuál es el propósito del editor?

   a) No.

   b) Un poco.

   c) Moderadamente.

   d) Mucho.

   e) Completamente.

   ¿Podría resumir brevemente dicho propósito?

6. ¿El editor ha satisfecho sus expectativas en general?

   a) No.

   b) Un poco.

   c) Moderadamente.

   d) Mucho.

   e) Completamente.

   En caso negativo, ¿por qué no se han satisfecho?

7. ¿Cuáles son las funcionalidades que el editor proporciona en cuanto a los dominios CEP? (se permiten múltiples respuestas)

   a) Definir gráficamente un nuevo dominio.

   b) Autodetectar un dominio (a partir de la información que fluya por una SOA 2.0 con la que el editor esté conectado).

   c) Validar y guardar un modelo de dominio.

   d) Modificar y eliminar un modelo de dominio existente.

   e) Personalizar con imágenes un dominio.

   f) Importar y exportar un modelo de dominio.

   g) Otras (especifique cuáles):

8. ¿Cuáles son las funcionalidades que el editor proporciona en cuanto a los patrones de eventos? (se permiten múltiples respuestas)

   a) Definir gráficamente nuevos patrones de eventos.

   b) Ahorrar tiempo al definir los patrones gráficamente (debe tenerse en cuenta que si se implementasen mediante un lenguaje de procesamiento de eventos, el usuario tendría que conocer previamente dicho lenguaje).

   c) Generar el código de los patrones de eventos modelados.

   d) Validar y guardar los modelos de patrones de eventos.

   e) Duplicar, modificar y eliminar modelos de patrones de eventos existentes.

   f) Personalizar con imágenes los patrones de eventos.

   g) Importar y exportar los modelos de patrones de eventos.

   h) Notificar los eventos complejos (situaciones críticas o relevantes) a los usuarios interesados, por ejemplo, a través de servicios de correo electrónico y de redes sociales.

   i) Otras (especifique cuáles):

9. ¿Considera útiles las funcionalidades proporcionadas por el editor?

   a) No.

   b) Sí.

   En caso negativo, ¿qué funcionalidades podrían añadirse?

10. ¿Considera irrelevantes algunas funcionalidades del editor?

    a) No.

    b) Sí.

    En caso afirmativo, especifique cuáles y explique los motivos:

11. ¿Considera irrelevantes algunas herramientas de la paleta gráfica del editor?

    a) No.

    b) Sí.

    En caso afirmativo, especifique cuáles y explique los motivos:

12. ¿Ha podido crear correctamente con el editor los patrones de eventos que se le han pedido para un dominio concreto?

a) No.

b) Un poco.

c) Moderadamente.

d) Mucho.

e) Completamente.

En caso negativo, ¿cuáles son los problemas que ha encontrado?

13. Una vez definidos los patrones de eventos gráficamente, ¿tiene claro cuál es el propósito de dichos patrones?

    a) No.

    b) Un poco.

    c) Moderadamente.

    d) Mucho.

    e) Completamente.

    En caso negativo, ¿qué aspectos del diseño gráfico no se comprenden?

14. ¿Qué nivel de destreza en lenguajes de procesamiento de eventos considera que se requiere para definir los patrones de eventos gráficamente?

    a) Novato.

    b) Principiante.

    c) Competente.

    d) Avanzado.

    e) Experto.

15. ¿Cree que el editor podría reducir el tiempo que necesita para definir los patrones de eventos; en lugar de escribirlos *a mano*, codificándolos mediante un lenguaje específico de procesamiento de eventos?

    a) No.

    b) Un poco.

    c) Moderadamente.

    d) Mucho.

    e) Completamente.

16. ¿Le ha sido fácil encontrar cada opción en el editor para crear los patrones de eventos gráficamente?

a) No.
b) Un poco.
c) Moderadamente.
d) Mucho.
e) Completamente.

17. ¿La tarea que se le ha propuesto realizar con el editor se ha comprendido fácilmente?

a) No.
b) Un poco.
c) Moderadamente.
d) Mucho.
e) Completamente.

En caso negativo, ¿por qué no? Indique mejoras para describir la tarea:

18. ¿Cuánto tiempo (en minutos) ha empleado para llevar a cabo la tarea propuesta?
19. ¿Tiene alguna sugerencia para mejorar el editor?

a) No.
b) Sí.

En caso afirmativo, especifique cuáles: